UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

MANUAL DE USUARIO

"TUTOR PARA LA ENSEÑANZA BASICA DE LA PROGRAMACION ORIENTADA A OBJETOS EN AMBIENTE SMALLTALK/V"

P R E S E N T A N :

ALBERTO COLMENARES

EDUARDO MONTUFAR FARFAN

ASESOR: ING. JUAN JOSE CARREON GRANADOS

CIUDAD UNIVERSITARIA, D. F. 1995

MANUAL

DE

USUARIO

Tabla de contenido

Introducción.

El presente tutor se emplea para la enseñanza básica de la programación orientada a objetos dentro del ambiente Smalltalk/V. Explicando y exponiendo los conceptos más básicos, que permitan al alumno lograr obtener una base sólida y extensa acerca de esta nueva metodologia, que le permitirá entender la mayoría de las herramientas poderosas y amigables, que ofrece el lenguaje.

Este tutor ha sido diseñado tomando en cuenta las necesidades y problemas del alumno al tratar de aprender un medio ambiente nuevo de programación. Además, el sistema ha sido concebido conteniendo las recomendaciones de personas expertas en el medio, basándose en la bibliografía de los mismos. Se ha puesto gran empeño en el diseño de este sistema para proporcionar una herramienta versátil y flexible, que permita al alumno un aprendizaje razonable, dándole bases para que posteriormente pueda introducirse en aspectos avanzados del lenguaje, es decir, una vez comprendido este tutor podrá ejecutar el ambiente natural de Smalltalk en el cual esperamos se sienta familiarizado.

CAPITULO 1 Descripción general del Tutor

Instalación del Tutor.

En el diskette que se proporciona se encuentran los archivos necesarios para la ejecución del tutor. Cuidando que el alumno poco experimentado no tenga problemas, se ha creado un archivo de procesamiento por lotes llamado "INSTALLT .BAT", para que pueda pasar tales archivos a el disco duro de la computadora, de forma rápida. Dicho archivo crea el directorio "TUTOR" dentro de el directorio "VWIN" que a su vez contiene los archivos de el ambiente de programación de Smalltalk/V, sin embargo, si ya existen los archivos del directorio "TUTOR", el programa no los instalará nuevamente; que aunque no son muchas las modificaciones que se le pueden hacer a los archivos durante su uso, si es importante que no se pierdan los resultados de los ejemplos desarrollados. En el directorio "TUTOR" se colocarán todos los archivos concernientes a el presente trabajo, los cuales están integrados tanto por cada uno de los capítulos de los cuales consta el tutor, como por los archivos que contienen algunos ejemplos que serán utilizados en el transcurso de el aprendizaje.

Elementos del Tutor.

El tutor se compone de el presente manual, cinco archivos contenidos en un directorio llamado tutor. Dichos archivos contienen el código de los ejemplos que el alumno debe ejecutar dentro de el ambiente Smalltalk/V, basado en la lectura del manual. Se espera que con esta combinación el alumno logre un aprendizaje efectivo.

Se recomienda leer el manual, a la vez que se ejecutan los ejercicios del tutor sobre la computadora. Los diversos archivos que contienen los ejemplos están divididos por capítulos, equivalentes a los capítulos explicados en el manual. Asi, el Capítulo 2: Introducción a la Interfaz gráfica de Smalltalk/V, le corresponde el archivo capitulo 2 que contiene los ejercicios respectivos.

Antes que se pueda hacer cualquier cosa, es necesario iniciar el tutor. Para lograr esto, sigue los siguientes pasos :

- Ejecuta Windows.
- Accesa el Grupo que se llama Smalltalk/V Windows 2.0, y busca el icono con el rótulo Smalltal/V Windows 2.0. Haz doble click sobre éste icono, donde entrarás a una pantalla de inicio típica del ambiente.
- Habiendo realizado lo anterior, ya te encuentras en el ambiente de programación de Smalltalk/V, y estás en posibilidad de ejecutar comandos (sin embargo, por el momento debes esperar).

- Para cargar al ambiente de cualquier capítulo debes activar la opción **File** en la barra de menús. Aparecerá un menú de persiana donde selecciona **Open...** . A continuación aparece un cuadro de diálogo, busca el directorio tutor y accesa su contenido; ahora elige el archivo del capítulo que desees abrir. Deberá aparecer una nueva ventana en Smalltalk/V mostrándote el código de los ejemplos.

La notación utilizada en el manual es la siguiente :

- Los nombres propios de clases, variables, métodos, instancias y algunas otras estructuras importantes de Smalltalk/V serán impresas en negrillas (**BoldFace**). Por ejemplo, utilizando contraste en : "class**Dictionary**". Nombres propios de otros productos de software se mostrarán en letras mayúsculas.

- Los ejemplos de código están separados del texto y serán impresos en negrilla. Algunos espacios están insertados entre argumentos para facilitar su lectura.

- Las funciones y las sentencias serán indentadas a diferentes espacios por nivel, como se muestra a continuación:

```
|i < 10|
        whileTrue:[
        sum:= sum + (a at: 1)
        i := i + 1]
```

- Los titulos de los capítulos son escritos en negrilla, a su vez los subtítulos también.

Requerimientos del Sistema.

El tutor de Smalltalk/V para windows requerirá un procesador 80286, 80386, 80486 o 80586 en una computadora capaz de correr windows 3.00 y versiones posteriores, teniendo:

- Un mínimo de 3 Megabytes en RAM.
- Un disco duro de al menos 40 Megabytes.

Los siguientes elementos son opcionales pero mejoran sustancialmente el rendimiento del sistema:

- Memoria RAM adicional (para ciertas aplicaciones).
- Un mouse.
- Un coprocesador matemático (80287, 80387 o 80487).

Contenido del presente manual

El capítulo 2 muestra los conceptos básicos y generales referentes a la interfaz gráfica, abarcando el manejo de las diversas ventanas disponibles que componen el ambiente. Así mismo, se trabajan los diversos elementos que componen una ventana, por ejemplo botones de maximización, de minimización, de colapso, barras de etiquetas, barras de menú y bordes para el desplazamiento del tamaño de la ventana. Finalmente se describe un apartado titulado viajando a través de Smalltalk/V donde se describen brevemente todas las ventanas del ambiente y su propósito general.

El capítulo 3 nos introduce a la definición básica de lo que son los objetos, mensajes y estructuras de control. Se utilizan ejemplos sencillos y didácticos con el objetivo de cubrir un aprendizaje fácil y ameno. Dentro de las estructuras de control se cubren las más importantes tratando de ser congruentes con otros lenguajes de programación.

El capítulo 4 explica de manera general las Clases y los Métodos; la manera de crearlos, modificarlos e inclusive eliminarlos. Mediante el desarrollo de un ejemplo se crea una clase y algunos métodos asociados a está para utilizar la ventana Class Hierarchy Browser ya explicada en el capítulo 2. También se explica una de las características de la programación orientada a objetos: la Herencia. Se explica la manera en que esta propiedad es usada en el ambiente.

El capítulo 5 aborda los Flujos (Streams), las Colecciones y el uso del Depurador. Se explican los Flujos más importantes, en el manejo de archivos, cadenas y arreglos. En cuanto a las Colecciones se observan las propiedades que hacen diferentes a los Diccionarios, las Bolsas y los Conjuntos. Se desarrolla un ejemplo que de manera deliberada contiene errores de forma que al tratar de aplicarlo al sistema se aprenda a utilizar el Depurador, herramienta imprescindible en la programación Smalltalk/V.

En el capítulo 6 se describen las herramientas gráficas disponibles en Smalltalk/V.Como sabemos son una parte importante en cualquier lenguaje de programación. Por esto, éste capítulo inicia con la definición más básica que nos permite conformar un gráfico, es decir, los pixeles en pantalla y el mapeo de bits, por medio de los cuales es posible representar cualquier tipo de imagen. Se dan ejemplos concretos, interactúan usuario y sistema a través de vidrios gráficos llamados prompter o vidrios para interacción.

Alcances y limitaciones del presente manual.

Para hablar acerca del alcance de este manual, se deben considerar varios aspectos. Uno de ellos es que no es una labor sencilla enseñar una metodología de programación al tratarse de un lenguaje. Por tal motivo y con toda honestidad el manual trata de alcanzar un proceso de enseñanza que resulte al alumno lo más libre de complicaciones en medida de no sacrificar una enseñanza consistente.

Otro aspecto, se considera al evaluar la cantidad de material con que cuenta Smalltalk/V en si mismo, pues el lenguaje es muy extenso.

En concreto los alcances y límites del manual los encontramos en la visión general y básica que se expone en este tutor.

CAPITULO 2 Introducción a la interfaz gráfica de Smalltalk/V

Iniciando Smalltalk/V

Antes de poder hacer cualquier cosa en Smalltalk/V, el ambiente de programación de Smalltalk/V debe ser activado haciendo doble click con el mouse sobre el icono de grupo que representa al ambiente Smalltalk/V, dentro de Windows. Para hacer esto debes primero entrar al ambiente Windows y después localizar en el grupo de programas, el grupo que representa al ambiente Smalltalk/V. Ve al administrador de programas y activa el icono de Smalltalk/V. Debes obtener una figura similar a la siguiente:

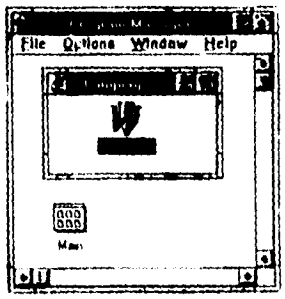


Figura 2.1 Después de hacer doble-click en el administrador de programas sobre Smalltalk/V.

Una vez que haz localizado el icono de Smalltalk/V, haz doble click sobre él para conseguir entrar e iniciar el ambiente de programación de smalltalk/V. La figura siguiente ilustra una ventana de inicio típica.

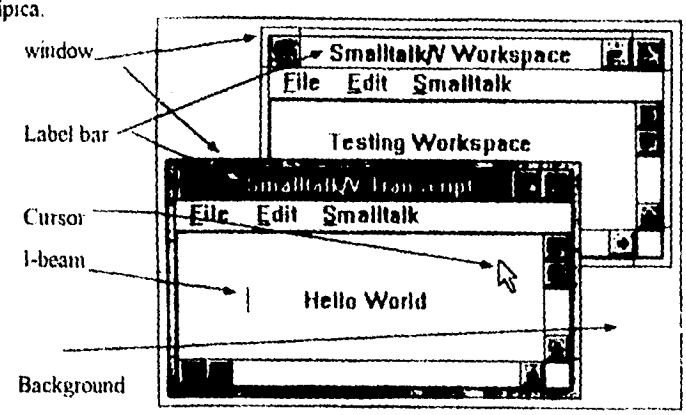


Figura 2. 2 Ventana clásica de inicio

Como se muestra en la figura 2.2 , el ambiente de ventanas de Smalltalk/V al ser activado muestra varios objetos; en este caso, un cursor del mouse, dos ventanas con barras de etiqueta (labels bars), un fondo entre las ventanas (background), un I-beam y un cursor de texto. Cada uno de estos componentes se explica a continuación :

- Una ventana **(window)** es una área sobre la pantalla limitada por un límite especial. En la figura 2.2 , dos ventanas rectangulares se muestran: una ventana transcript y una ventana workspace. Como puedes comprobar las ventanas pueden ser desplazadas sobre la pantalla y también puedes maximizarlas o minimizarlas. Si observas la figura 2.2 te darás cuenta que puedes anteponer una ventana sobre otra. En la figura 2.2 la ventana transcript se antepone a la ventana workspace.

- Una barra de etiqueta **(label bar)** es una área en la parte superior de una ventana que contiene un título.
- El **background** es el área entre las ventanas.
- Un **I-beam** es una línea vertical dentro de una ventana de texto (una ventana que puede aceptar y desplegar texto). El I-beam marca el inicio donde el texto será insertado o borrado. Nota que el I-beam se puede confundir con el cursor de texto, pero con la práctica será simple distinguirlos. Por ejemplo mueve el mouse y se moverá el cursor de texto, pero el I-beam no.

- Un cursor es un **icono** especial (un dibujo) que indica la localización del mouse sobre la pantalla.. Cada representación del icono muestra que se esta realizando algún proceso o que esta disponible para ejecutarse. Por ejemplo el cursor de texto indica que el mouse esta sobre la porción de la pantalla donde el texto puede ser escrito, mientras un cursor en forma de flecha, que puede aparecer sobre un componente como la barra de etiqueta indicando que texto no puede ser escrito en esa posición. Un cursor de ejecución muestra que la computadora esta ejecutando algún proceso. Un cursor en forma de cruz indica que un usuario esta realizando alguna selección. Y finalmente los cursores para modificar el tamaño de las ventanas indican que las esquinas de una ventana pueden ser movidas. El conjunto completo de iconos que representan cursores, se muestra en la siguiente figura:

Figura 2. 3 Conjunto completo de iconos para todos los iconos.

Claramente, cuando el cursor del mouse se mueve, se asocia sobre la pantalla que contiene los componentes de una ventana. La asociación entre el mouse y estos cursores es obvio que es conveniente y podemos omitir hablar de un cursor especifico. Y precisamente, haciendo click o doble click con el botón del mouse sobre una área de la pantalla es una forma breve de primero mover el cursor a la área especifica y entonces presionar el mouse para realizar alguna actividad.

Salvando la imagen.

Cuando se trabaja con Smalltalk/V, todo lo que se hace es guardado en dos archivos especiales:

- La imagen, un archivo llamado **v. exe** que contiene el estado actual de tú ambiente de programación (este archivo no puede ser impreso).
- El archivo cambios, llamado **change. log** que contiene todas las instrucciones que tú ejecutaste mientras trabajabas dentro de Smalltalk/V. (este archivo puede ser impreso).

El estado actual del ambiente de Smalltalk/V puede ser salvado en cualquier momento. Si tú realizas cambios al sistema (escribiendo dentro de las ventanas) y salvas la imagen, la próxima vez que inicies Smalltalk/V, iniciará con los cambios que realizaste. Por otra parte, tú puedes hacer cambios al sistema y deliberadamente elegir no salvar la imagen. La próxima vez que inicies Smalltalk/V, te encontrarás en el estado anterior con el que habias iniciado originalmente. Para salvar la imagen sigue los siguientes pasos:

- Mueve el cursor al menú **File**, como se muestra en la figura 2. 4 y sostén presionado el botón izquierdo del mouse. Un menú de persiana aparecerá, como se muestra en la figura 2.5.
- Mueve el cursor al comando **Save Image...** y libera el botón del mouse. La caja de diálogo que se muestra en la figura 2.6 aparecerá, preguntando si tú deseas salvar la imagen.

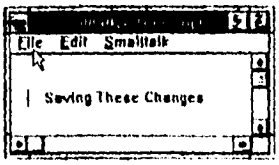


Figura 2. 4 Ventana Transcript con el mouse sobre el menú File (archivo)

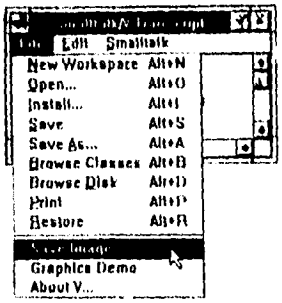


Figura 2. 5 Selección de la opción "Save Imagen" en el menú de persiana desplegable

- Haz click sobre la caja **Make Backup** si estás seguro de querer hacer un respaldo. El Backup puede usarse para sustituir al original si vas a destruirlo. El backup de **v. exe** es llamado **v. back**. Generalmente, si estás trabajando con un disco duro, deberías hacer un backup.
- Finalmente, haz click sobre **yes**. Alternativamente puedes hacer click sobre **no** si cambias de idea y no deseas salvar la imagen.

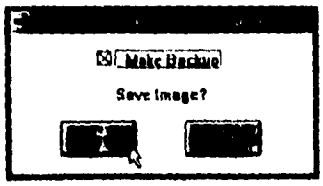


Figura 2. 6 Responde "Yes" a la caja de diálogo.

Saliendo de Smalltalk/V

Cuando tú terminas de trabajar con Smalltalk/V, debes salir de su ambiente. Sigue los siguientes pasos:

- Mueve el cursor a la esquina superior izquierda de la ventana Transcript (a la caja de cerrado), como se muestran en la figura 2.7 y presiona el botón izquierdo del mouse. Un menú de persiana aparecerá, como se muestra en la figura 2.8.
- Mueve el cursor sobre el menú al comando **Exit Smalltalk/V...** y libera el botón del mouse. La caja de diálogo mostrada en la figura 2.9 aparecerá preguntando si deseas salvar la imagen.

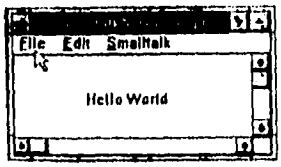


Figura 2.7 Ventana Transcript con el cursor sobre el menú File para poder activar la opción cerrar.

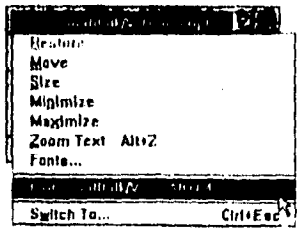


Figura 2. 8 Selecciona "exit Smalltalk/V" en el menú desplegable para salir del ambiente.

- Si deseas salvar la imagen, te recomendamos que elijas **Cancel** y salves la imagen explícitamente como se describió en la sección previa. Si tú salvas mientras estas saliendo de Smalltalk/V y ocurre un error (porque hay insuficiente espacio en disco), no habrá oportunidad de borrar algunos archivos para conseguir espacio e intentar salvar de nuevo. Y si tú ya tienes salvada la imagen, entonces debes elegir **No** (no es necesario que salves otra vez).

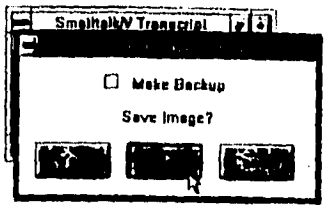


Figura 2. 9 Respondiendo "No" a la caja de diálogo.

Exploración de los componentes

Un programador experimentado con Smalltalk/V conocerá todos los nombres para los componentes de las ventanas, será capaz de manipular los menús asociados con las ventanas, y será capaz de desplazarse a través de las ventanas. Esta sección esta dedicada a darte toda esta información en una forma condensada. No es necesario que intentes recordar todo esto a la primera lectura. Nosotros tenemos la oportunidad de aprender esta información a través de la práctica y en la exposición dentro de contextos cuando ello sea necesario. La meta para esta primera sección es introducirte a la terminología, de tal forma que estos términos te suenen familiares al usarlos posteriormente.

Componentes de las Ventanas

Las ventanas no sólo son dibujos (ver figura 2.10). Ellas tienen partes, tales como botones donde se puede hacer click, por ejemplo, para cambiar el tamaño de la ventana y barras de desplazamiento que causan que el texto se mueva horizontal o verticalmente. Las ventanas generalmente consisten de una barra de etiqueta, un borde, uno o más vidrios o subventanas (ver figura 2.10), y botones especiales para manipular la ventana. El botón de una ventana es activado haciendo click con el botón del mouse cuando el cursor esta encima de algún botón en la ventana.

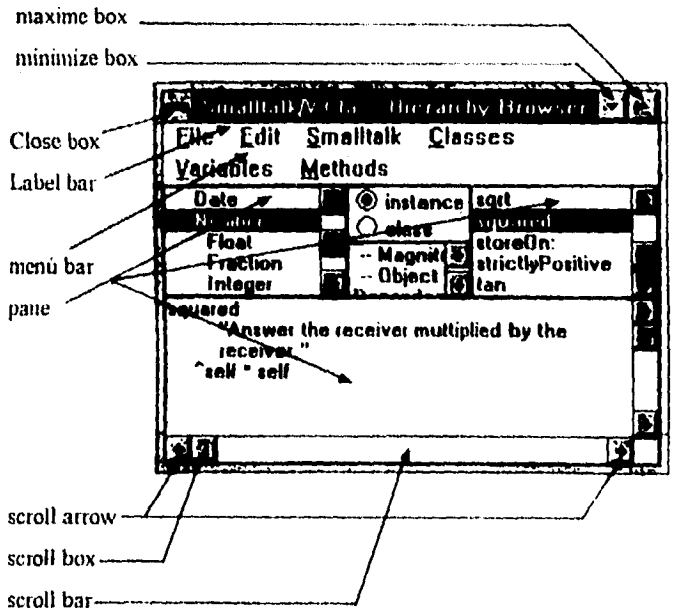
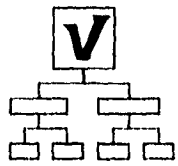


Figura 2.10 Ventana con tres vidrios

Una breve definición de cada parte de una ventana se da a continuación. Refiérete a los diagramas para que identifiques todas las partes apropiadamente.

- **La caja de maximización** es un botón que causa que la ventana se agrande a su máximo tamaño posible cuando tú haces click sobre él con el botón izquierdo del mouse. Para restaurar la ventana a su tamaño inicial, haz click sobre la caja de maximización nuevamente.

- **La caja de minimización** es un botón que causa que la ventana decrezca a su mínimo tamaño cuando tú haces click sobre él con el botón izquierdo del mouse. Observa que la ventana se transforma en un icono (ve figura 2.11). Como es obvio, diferentes tipos de ventanas, generan diferentes iconos. Para restaurar la ventana haz doble click sobre el icono o sobre la etiqueta del icono.



Smalltalk/V Class Hierarchy Browser

Figura 2.11 Muestra la ventana minimizada de la figura 1.10

- **La caja de cerrado** es un botón que causa que la ventana desaparezca del ambiente cuando tú haces click sobre él con el botón izquierdo del mouse, como es obvio se usa para cerrar las ventanas.
- **La barra de etiqueta** es una área rectangular sobre la parte superior de la ventana que contiene texto, llamado etiquetas o títulos.
- **La barra de menú** es una área rectangular debajo de la barra de etiqueta que contiene una lista de los elementos del respectivo menú. Un menú de persiana aparece debajo de cada elemento de la barra de menú cuando tú presionas el botón izquierdo del mouse sobre alguna opción elegida (como se muestra en la figura 2.12). Cuando la ventana es lo suficientemente ancha para contener todos los elementos del menú, ellos aparecen en una sola línea; de otra manera ellos aparecen en una segunda línea como se muestra en las figuras 2.10 y 2.12.
- **Un vidrio** es una subventana sin una barra de etiqueta o botones de ventana asociados. Este puede ser una área para editar texto, o para desplazar una lista de opciones
- **La barra de desplazamiento** es un rectángulo que indica que los datos de un vidrio pueden ser movidos horizontal o verticalmente. El desplazamiento puede ser logrado en tres formas: haciendo click sobre un **botón flecha**, que se encuentran al principio y al final de la barra de desplazamiento; haciendo click sobre el botón de desplazamiento, un pequeño cuadrado que puede ser desplazado a través de la barra de desplazamiento; o haciendo click con el mouse dentro de cualquier parte en la barra de desplazamiento.

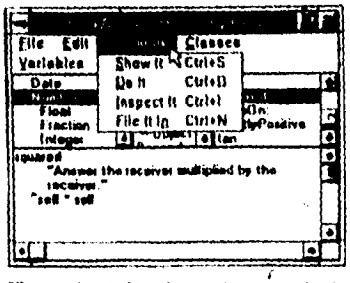


Figura 2. 12 Presionando el botón izquierdo del mouse sobre la opción Smalltalk del menú

Viajando a través de los menús

Generalmente, cada vidrio en una ventana tiene una labor especial para lo cual fue diseñado. En consecuencia, cada vidrio tiene un menú de comandos que pueden ser seleccionados por el programador. Estos menús de comandos están asociados con elementos específicos de la barra de menú. Los comandos de cada elemento pueden aparecer haciendo click con el botón izquierdo del mouse sobre algún elemento específico. Cuando esto se hace, el menú del comando aparece inmediatamente debajo del elemento (ellos son llamados menús de persianas. Cuando hay varios vidrios en una ventana, los comandos se aplican en el vidrio activo) del vidrio donde hiciste click con el mouse. La figura 2.13 muestra como se puede elegir un comando sobre la barra de menú para ejecutar la expresión "1+2". Nota que primero se escribió el texto, para ser seleccionado posteriormente y después elegimos de la barra de menú la opción **Show It** para conseguir ejecutar la expresión anterior. Esto se estudiará con mayor detalle en apartados posteriores. Por ahora, lo que nos concierne es la mecánica para optener y seleccionar comandos de la barra de menú. Los pasos necesarios para esto, se describen a continuación:

- Sostén presionado el botón izquierdo del mouse sobre la palabra **Smalltalk** en la barra de menú (ver figura 2.13). Esto causa que aparezca el menú de persianas conteniendo varios comandos.
- Aún presionando el botón del mouse, mueve el cursor hacia abajo al comando deseado (por ejemplo, el comando **Show It**) y entonces libera el botón del mouse. Si cambias de idea, simplemente haz click fuera del menú de persiana.

Altematívamente, es posible liberar el menú de persiana sin mantener presionado el botón del mouse mientras te mueves al comando deseado. Para lograrlo sigue los siguientes pasos:

- Haz click sobre la palabra **Smalltalk** para desplegar el menú de persiana
- Mueve el cursor hacia abajo al comando deseado sin presionar el botón del mouse
- Selecciona un comando haciendo click con el mouse sobre la elección deseada.

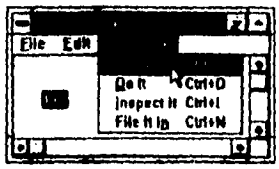


Figura 2. 13 Selección de un vidrio en el menú desplegable.

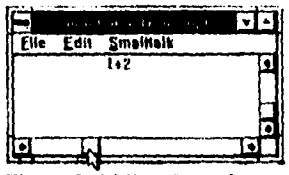


Figura 2. 14 Desplazamiento a la derecha con el botón del fondo.

Desplazamiento a través de las ventanas.

Es posible desplazarse en la mayoría de las ventanas que existen en Smalltalk/V. Un ejemplo es la ventana Transcript que contiene un vidrio de texto (**text pane**) donde podemos escribir. En general la posición de la caja de desplazamiento indica la parte de los datos visibles dentro de un vidrio. Por ejemplo, si la caja de desplazamiento vertical esta a la mitad, entonces el vidrio mostrará la parte de los datos que se encuentran a la mitad (en consecuencia, la parte de arriba y la del fondo no son visibles). Ver figura 2.14 que intenta explicar esto en forma gráfica.

Para los vidrios de texto, las barras de desplazamiento vertical y horizontal son permanentemente visibles y el desplazamiento se logra haciendo click con el mouse sobre la flecha de desplazamiento, la caja de desplazamiento o sobre el resto del área de la barra de desplazamiento. Además, el desplazamiento puede ser iniciado manteniendo presionado el botón del mouse sobre algún lugar en el vidrio y moviendo el mouse al lugar deseado. El vidrio se desplaza también para permitir que el usuario pueda seleccionar más texto y para cancelar la selección basta hacer click en cualquier parte del vidrio. En general, podemos desplazamos directamente manipulando la barra de desplazamiento o sus componentes, mientras haya texto visible. Si no hay nada escrito, no es posible continuar el desplazamiento.

Las diferentes ventanas de trabajo.

Una de las primeras cosas que tú debes hacer con un carro nuevo es sentarte dentro de él y tocar las diferentes partes para desarrollar un sentimiento de confianza por el carro. Lo mismo aplica a un nuevo ambiente de programación. Antes de que puedas sentirte confortable con el ambiente de Smalltalk/V, necesitamos sentir que tenemos control del mismo. El mejor camino para desarrollar nuestra confianza es jugar con el sistema, jugar con las diferentes ventanas.
Nuestra tarea en esta sección será desarrollar algo de experiencia sobre como crear ventanas, en particular una ventana llamada de nuevo trabajo **(workspace)**, a movernos alrededor de ellas, aprender a cambiar sus tamaños, y a cancelarlas si así lo deseamos.

Empezando

Comienza ejecutando Smalltalk/V como se indicó anteriormente. Tú ventana de inicio debe verse similar a la mostrada en la figura 2.2. Si no es así, no te preocupes. Aprenderemos rápidamente a cambiar las cosas para organizar la ventana que se requiera.

Creando una nueva ventana de trabajo (Workspace).

Nosotros podemos ignorar las ventanas que existen y crear nuevas. En particular, podemos crear un tipo especial de ventanas, llamadas ventanas de trabajo. Para hacer esto, sigue los siguientes pasos:

- Mantén presionado el botón izquierdo del mouse sobre la entrada **File** en la barra de menú. Aparecerá el menú de persiana, como se muestra en la figura 2.15.
- Del menú selecciona **New Workspace** moviendo el cursor hasta que este encima del comando y libera el botón del mouse. A continuación, el menú de persiana desaparecerá y aparecerá una nueva ventana de trabajo, como se muestra en la figura 2.16.

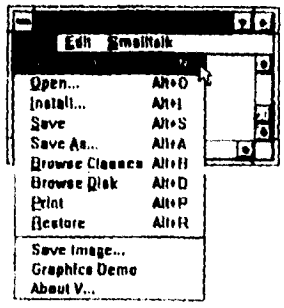


Figura 2. 15 Prioridad para crear una nueva ventana de trabajo (Workspace).

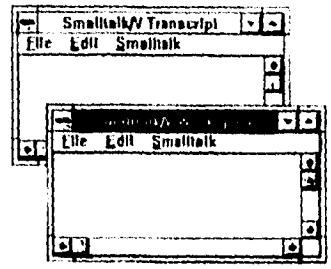


Figura 2. 16 Después de la creación de una ventana nueva de trabajo (Workspace).

Repite este procedimiento siempre que deseos crear dos o tres ventanas de trabajo. Este proceso de creación de una ventana es llamado **abriendo** una ventana.

Activación de ventanas.

No importa cuantas ventanas se encuentren abiertas en el ambiente Smalltalk/V, ya que sólo una ventana es reconocida por Smalltalk/V como activa a un mismo tiempo. Una ventana activa puede ser reconocida por el hecho de que la barra de etiqueta se encuentra especialmente resaltada (ver la ventana de trabajo en la figura 2.16). Una ventana inactiva se activa haciendo click con el mouse en cualquier parte dentro de la ventana. Si una ventana es parcialmente cancelada al generar otra nueva, la ventana cancelada se coloca en la parte superior de la nueva ventana. Experimenta con el sistema haciendo click sobre sucesivas ventanas.

Movimiento de ventanas

Las ventanas pueden moverse fácilmente manteniendo presionado el botón izquierdo del mouse sobre la barra de etiqueta de la ventana. Hecho esto, mueve el mouse a cualquier posición y la ventana se desplazará siguiendo el movimiento del mouse. Una vez movida la ventana a una posición deseada libera el botón del mouse.

Reajuste de ventanas

Reajustar una ventana significa cambiar su tamaño. Para lograr esto, sigue las indicaciones:

- Mueve el cursor a un borde o a una esquina de la ventana. El cursor del mouse actual cambiara a un cursor de ajuste, como se muestra en la figura 2.17. El tipo de cursor que se presente dependerá de la posición donde se coloque el cursor, pudiendo ser el borde de la ventana o una esquina.

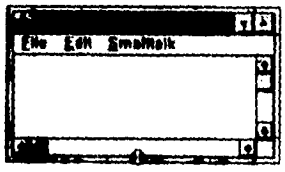


Figura 2.17 Ajustando una ventana
(nota el cursor de ajuste en el fondo)

- Presiona el botón izquierdo del mouse y mueve el mouse. El borde o la esquina de la ventana deberá seguir el movimiento de mouse. Cuando el borde o la esquina se han movido a la posición deseada, libera el botón izquierdo del mouse.

Cerrado de ventanas

Una ventana se cancela o se descarta cerrándola. Una ventana puede ser cerrada como sigue:

- Haciendo doble click sobre la caja de cerrado en la esquina superior izquierda de la ventana.

Alternativamente,

- Presionando sobre la caja de cerrado para obtener un menú de persiana.
- Haciendo click en la entrada close del menú.

En el ambiente de Smalltalk/V, todas las ventanas pueden ser cerradas. Se debe considerar, que al cerrar la ventana del Transcript es equivalente a salir de Smalltalk/V. No debe preocuparnos si accidentalmente ocurre esto, ya que una caja de diálogo aparecerá (ver figura 2.9) preguntando si deseas cancelar la operación.

Experimenta cerrando varias ventanas. Si fuera necesario abre algunas con la opción new workspace e inmediatamente ciérralas.

Colapsando ventanas

Cuando una ventana en Smalltalk/V es colapsada, únicamente un icono y la barra de etiqueta son visibles (ver figura 2.19). Una ventana se colapsa haciendo click sobre la caja de minimización

(la caja más a la izquierda en la esquina superior derecha de la ventana).

Para restaurar una ventana colapsada a su estado original, basta con hacer doble click sobre el icono o la barra de etiqueta.

Experimenta colapsando algunas ventanas. Abre algunas, si fuera necesario. Nota que las ventanas colapsadas se pueden desplazar igual que las ventanas no colapsadas.

Agrandado (Zooming) de ventanas

Cuando una ventana contiene texto que debe ser leido o cambiado, es conveniente hacer que la ventana sea lo bastante grande y cubra toda la pantalla. Una forma de hacer esto, es usar la opción de **Maximizar**. Para hacer este proceso simple, una ventana puede ser agrandada, haciendo click sobre la caja de maximización (la caja más a la derecha en la esquina superior derecha de la ventana). Para restaurar la ventana, basta hacer click nuevamente sobre la caja de maximización usada anteriormente.

Un viaje a través de las diferentes ventanas de Smalltalk/V

Existen diferentes tipos de ventanas en el sistema. Cada tipo de ventana tiene un propósito especial y una apariencia particular. En capitulos posteriores, consideraremos a detalle las diferentes ventanas. Por el momento, es suficiente conocer algo breve acerca de ellas. Las figuras que siguen, a partir de la 2.18 a la 2.23 ilustran ventanas cuyo propósito no lo explicaremos ahora. Las listamos para una referencia posterior.

- Una ventana de trabajo (**workspace**) es una ventana de texto usada por el programador para desplegar y mostrar información textual.
- Una ventana **Transcript** es una ventana workspace que puede ser usada por el sistema para mostrar mensajes dirigidos al programador.
- La clase de **Hierarchy Browser** es una ventana usada por el programador para ver la libreria de clases y para aumentar o modificar esta libreria de clases.
- La **disk browser** es una ventana usada por el programador para ver y modificar archivos de discos.
- La ventana **inspector** es una ventana usada para ver y modificar objetos dentro del sistema.
- La ventana **Walkback** aparece cuando ocurre un error de ejecución. Tales ventanas serán discutidas detenidamente en apartados posteriores.
- La ventana **debugger** es usada por el programador para investigar y corregir un error de ejecución. Los debugger son investigados posteriormente.

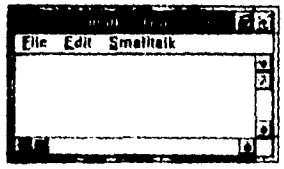


Figura 2.18 Ventana Transcript.

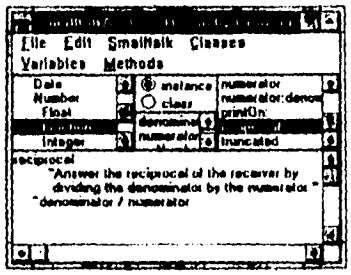


Figura 2.19 Una clase de la jerarquía browser.

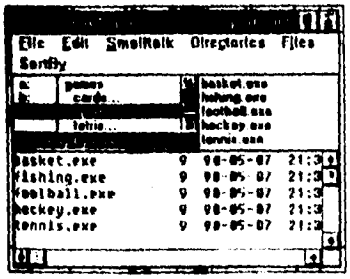


Figura 2. 20 Disco Browser

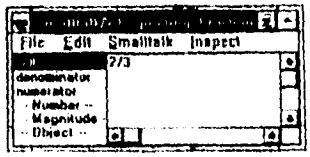


Figura 2. 21 Ventana Inspector

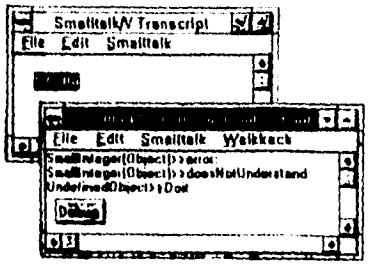


Figura 2. 22 Ventana "Walkback"
(De respuesta)

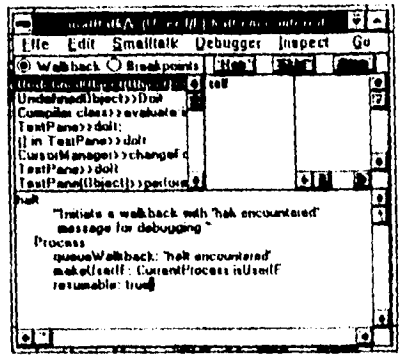


Figura 2. 23 Ventana Debugger
(Depuración)

Una forma simple de hacer uso de la programación en Smalltalk/V y del ambiente gráfico, es desempeñando funciones como una calculadora simple a través de expresiones sencillas como las siguientes:

1 + 2

1 + 2 + 3 + 4

1*2 + 3*4

Varias preguntas surgen espontáneamente,

- ¿ Exactamente cómo conseguimos las respuestas de las expresiones anteriores ?
- ¿ Cómo se calcula realmente la respuesta; es

  1*2 + 3*4 = (1*2) + (3*4) = 2 + 12 = 14 ó es

  1*2 + 3*4 = ((1*2) + 3)*4 = (2 + 3) *4 = 5*4 = 20 ?
- ¿ Cúan rico es el conjunto de instrucciones que nosotros podemos usar ?.

Un camino para responder a todas estas preguntas sería intentar evaluar cada una de las expresiones y contemplar sus resultados respectivos. Desafortunadamente no tenemos suficiente experiencia en el ambiente de Smalltalk/V para ser capaces de realizar los cálculos anteriores. Asi, por el momento debemos desviarnos un momento para conseguir entender mejor la ventana del Transcript, donde tales expresiones pueden ser evaluadas.

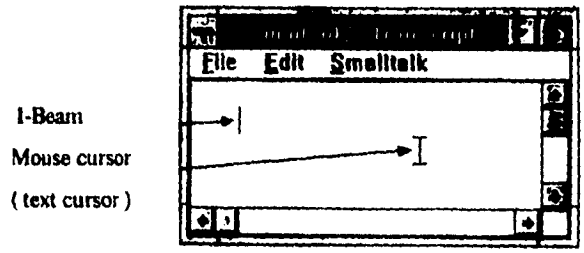


Figura 2.24 Ventana Transcript

Mecánica de la ventana del Transcript.

En el ambiente de Smalltalk/V, el texto puede ser escrito en diferentes tipos de ventanas; ejemplo la ventana del Transcript, ventanas de espacio nuevo de trabajo (workspace windows), o en el fondo de los vidrios de las ventanas browser. Siendo estrictos todas estas ventanas son capaces de evaluar texto, motivo por el cual nos concentraremos en la ventana del Transcript.

La ventana del Transcript es única porque siempre esta abierta en el ambiente de Smalltalk/V. Más aún, solo existe una única ventana de Transcript. Nuestra meta inmediata es aprender enseguida a usar la ventana del Transcript como una máquina de escribir.

Trabajando con texto.

Podemos empezar jugando con el **I-beam** dentro de la ventana del Transcript. El I-Beam es una línea vertical dentro de una ventana de texto. Marca el inicio donde el texto será insertado o borrado. En contraste la posición del mouse es representado por un cursor de texto. Como vemos en la figura 2.24 son muy similares. El cursor de texto se moverá siempre que el cursor del mouse sea movido; el I-beam por el contrario, no se moverá hasta hacer click con el mouse.
Usando el mouse, haz click en cualquier parte de la ventana Trascript. El I-beam desaparecerá de su posición actual y reaparecerá en la posición del cursor del mouse.

Escribiendo y borrando caracteres

Intenta escribir **hola** después de posicionar el I-beam en el centro del Transcript. Nota que el I-beam se mueve cada vez que un carácter es escrito. Además la tecla de retroceso (backspace) sirve para borrar caracteres. Observa también que la tecla suprimir (del) elimina caracteres a la derecha del cursor. Pruébalo. En conclusión, la tecla de retroceso elimina caracteres a la izquierda del I-beam y la tecla suprimir elimina a la derecha. Utiliza la que más te agrade.

Seleccionando palabras.

Ahora comenzaremos a escribir palabras con espacios en blanco. Los espacios en blanco se consiguen al presionar la tecla de espaciado (spacebar). Esta es la tecla más larga sobre el teclado.
Escribe ahora " hola grandes amigos". Nota que sucede cuando posicionas el apuntador del mouse sobre la palabra amigos y das doble click con el botón izquierdo del mouse. La palabra amigos se resalta. Si haces doble click sobre otra palabra, amigos se normaliza y ahora la nueva palabra esta resaltada. De esta forma seleccionas palabras para poder operar sobre ellas. Si deseas seleccionar una sola línea haz click al inicio de la misma. Para seleccionar todo un bloque, haz click con el mouse al inicio y manteniendo presionado el botón del mouse arrástralo hasta el final del bloque liberando el botón. Para cancelar la selección simplemente haz click en cualquier parte de la ventana.

Reemplazando texto

Supongase que necesitamos cambiar una parte de la siguiente expresión: " Hola, espero te encuentres bien ". Digamos que queremos cambiar la parte "espero te encuentres bien" por "ojalá estes agusto". Dos formas son posibles:

Reemplazando por otro texto nuevo.

Para reemplazar "espero te encuentres bien" por "ojalá estes agusto", primero seleccionamos "espero te encuentres bien" y entonces escribimos el texto nuevo "ojalá estes agusto. No es necesario presionar la tecla suprimir (del).

Reemplazando por supresión.

Para remover "espero te encuentres bien" primero seleccionalo, y después aprieta la tecla de retroceso (Backspace) o la tecla suprimir (del). En este momento puedes teclear "ojala estes agusto".

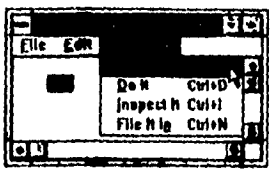


Figura 2. 25 Ventana Smalltalk obtenida sobre el menú en la ventana Transcript

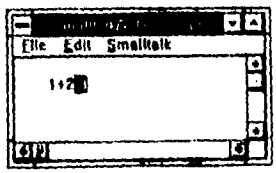


Figura 2. 26 Después de elegir "Show It"

Ejecución de simples comandos

Ahora estamos listos para evaluar algunas cosas simples. Escribe " 1+2 " en alguna parte de la ventana Transcript, seleccionalo y entonces elige **show it** en el menú **Smalltalk**, como se explica a continuación:

- presiona el botón izquierdo del mouse sobre la palabra Smalltalk en la barra de menú. Aparecerá un menú de persiana (figura 2.25). Este menú desaparecerá automáticamente cuando elijas una opción sobre de él.
- Para seleccionar un elemento del menú (en este caso el **show it**), mueve el cursor sobre el elemento deseado y libera el botón del mouse (el resultado se muestra en la figura 2. 26)

Ya estamos listos para experimentos más interesantes sobre la manera de trabajar del área de la ventana Transcript.

Experimentando para aprender.

La llave para el aprendizaje es la práctica directa. En esta sección nosotros adoptaremos esta idea para aprender los conceptos básicos de la programación en Smalltalk/V, asi como la terminología asociada.

Descubriendo acerca de los enteros

Ahora que conocemos como ejecutar "1+2" para conseguir "3",nos podemos preguntar que tan significantes son los espacios. ¿Obtendremos la misma respuesta si ejecutamos " 1 + 2" además de "1+2" ? Como podemos ver en las figura 2. 27 y 2. 28, parece que los espacios no tienen importancia. En concreto, veamos la figura 2. 29 y 2. 30 que aparentemente indican que podemos extender una expresión a través de múltiples líneas.

Pero debemos tener cuidado de no separar números muy grandes a través de varias líneas. Aparentemente tal separación de números es interpretada como dos números consecutivos sin un operador (+, -, etc) entre ellos. Ver figuras 2. 31 y 2. 32.

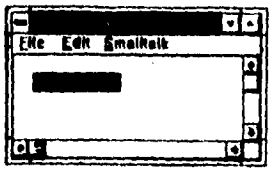


Figura 2. 27 Seleccionando una expresión con espacios de más.

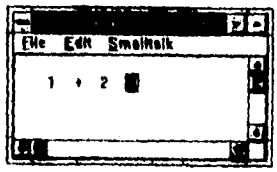


Figura 2. 28 Ejecutando la expresión que muestra que los espacios no son significantes

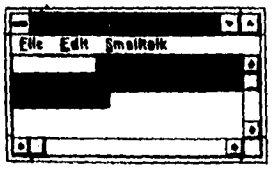


Figura 2. 29 Seleccionando una expresión extendida en varias líneas.

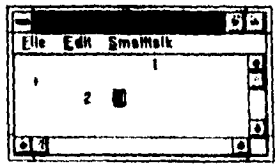


Figura 2. 30 Ejecutando la expresión extendida que se evalúa correctamente

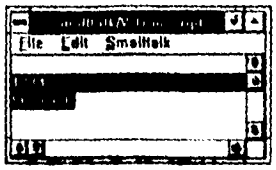


Figura 2. 31 Seleccionando una expresión donde el número se encuentra a través de líneas.

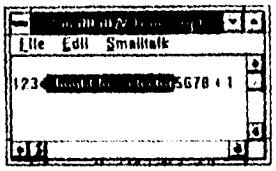


Figura 2. 32 Evaluando la expresión notamos que no es posible evaluar números escritos a través de líneas

Así, los espacios y los límites de las líneas son permitidos únicamente entre números y operadores. Ahora vamos a experimentar con algunos otros ejemplos. Para indicar que el resultado de la evaluación "1+2" es 3, debemos escribir:

1 + 2 = 3 (adición)

Considerando otros operadores:

4 - 1 = 3 (resta)

2 * 2 = 4 (multiplicación)

6 / 2 = 3 (división)

Algunas de las operaciones no existen en calculadoras estándar. Por ejemplo:

25 // 3 = 8 (división con truncación)

25 \\ 3 = 1 (calculo del resto)

En consecuencia, podemos usar expresiones más complejas.

1 + 1 + 1 = 3

5 - 1 - 1 - 1 = 2

2 * 2 * 2 = 8

12 / 3 / 2 = 2

¿ cúal es la expresión más simple que podemos escribir ?. ¿Se puede escribir tan sólo un número?. Claro que si. Ve las figuras 2. 33 y 2. 34.

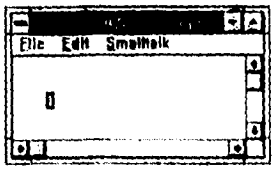


Figura 2. 33 Seleccionando "1" para evaluarlo.

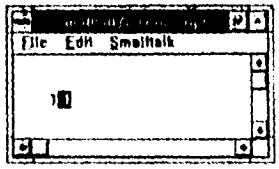


Figura 2. 34 Evaluando "1" aprendemos que los números se evalúan por si mismos.

En el ejemplo anterior, tan sólo hemos escrito un 1 que implica lo siguiente:

1 = 1

Y observa que ocurre el mismo caso con valores negativos. Evalúa el valor -1 y obtendrás lo siguiente:

-1 = -1

En consecuencia podemos escribir expresiones más complejas para números negativos:

4 - 6 = -2

2* -3 = -6

-6 / 2 = -3

Pero debemos tener cuidado. ¿Es "4-3" sin espacios o es 4 -3 (observa después del 4 un espacio en blanco y después -3) ?. Evalúa las dos formas:

a) 4-3 y

b) 4 -3

te darás cuenta que la forma correcta de escribir esta expresión numérica es la siguiente:

c) 4- 3 (espacio en blanco antes del 3)

d) 4 - 3 (espacio en blanco después del 4 y antes del 3).

Finalmente que pasa si mezclamos operadores. Por ejemplo, ¿ las siguientes expresiones dan el mismo resultado?

1 + 2 * 3

2 * 3 + 1

Al evaluarlas te darás cuenta de que no. Los resultados son diferentes.

1 + 2 * 3 = 9 y 2 * 3 + 1 = 7

Antes de explicar el porque de la diferencia, vamos a considerar algo de la terminología involucrada. En una expresión tal como " 1 + 2 " , el término + se llama **operador** , mientras 1 y 2 se llaman **operandos** del operador. El número 1 es el operando izquierdo del +, mientras el número 2 es el operando derecho del +. En una expresión que tenga dos operadores, " 1 + 2 * 3 ", es fácil identificar los operadores + y * pero sus operandos son más difíciles de identificar. La razón es que los operadores deben ser evaluados en algún orden y consecuentemente un operando debe ser el resultado de una evaluación previa. Por ejemplo, considera los dos posibles casos:

- si + es evaluado primero, "1+2*3" se simplifica a "3*3" que en turno se simplifica a 9. De aquí debería ser claro que el operando izquierdo del * no es 2, sino 3 (porque 1+2 = 3).

- si * es evaluado primero "1+2*3" se simplifica a "1+6" que en turno se simplifica a 7. En este caso, "2*3" o 6 debe ser el operando derecho del +.

Lo importante es darse cuenta que en Smalltalk/V el orden de evaluación es estrictamente de izquierda a derecha. El operador más a la izquierda es evaluado primero con sus operandos inmediatos y el resultado reemplaza ambos operandos y al operador; entonces el proceso es repetido hasta que no queden operandos en la expresión evaluada por Smalltalk. En consecuencia, las expresiones anteriores se evalúan de la siguiente forma:

1 + 2 * 3 = 3 * 3 = 9 y 2 * 3 + 1 = 6 + 1 = 7

Este proceso es más evidente cuando evaluamos una expresión más compleja. Considera la siguiente expresión. Vamos a subrayar la parte de la expresión que es evaluada en cada paso. En cada paso siguiente, el resultado sustituirá la parte subrayada :

1 + 2 * 3 -4 + 5 * 6 -7 -8 / 9 = ?
3 * 3 -4 + 5 * 6 -7 -8 / 9 = ?
9 -4 + 5 * 6 -7 -8 / 9 = ?
5 + 5 * 6 -7 -8 / 9 = ?
10 * 6 -7 -8 / 9 = ?
60 - 7 -8 / 9 = ?
53 - 8 / 9 = ?
45 / 9 = ?
5

¿ Pero que hacemos si necesitamos evaluar primero el operador " * " antes que el operador " + " en la expresión " 1+ 2 * 3 " ? . ¿ Hay algún camino para forzar un orden diferente de evaluación?. La respuesta es si. Podemos usar paréntesis para agrupar expresiones que son evaluadas como una unidad.

1 + (2 * 3) = 1 + 6 = 7 y (2 * 3) + 1 = 6 + 1 = 7

Como podemos ver el uso de paréntesis ha cambiado el orden de estas expresiones que ya habíamos evaluado anteriormente. Ahora, ¿ podemos usar paréntesis como se quiera ?. Por ejemplo, observa lo siguiente:

(1 + 2 + 3) (1) + 2
(((1 + 2) + 3) + 4) + 5 (((((1))))

Si, todas las expresiones anteriores son legales. En general, los paréntesis pueden rodear cualquier espresión que sirva como un operando para algún operador. Pero también, los paréntesis pueden ocasionar expresiones ilegales como las siguientes :

(1 + 2	falta paréntesis derecho
1 + 2)	falta paréntesis izquierdo
1)+(2	faltan paréntesis izquierdo y derecho
(1+) 2	el 1+ no puede ser usado como un operando

Si intentas evaluar una expresión ilegal, un mensage de error será generado. Por ejemplo, considera la figura 2. 35. Si esto sucede, usa la tecla de retroceso (backspace) para remover el error y continuar como si nada hubiera ocurrido.

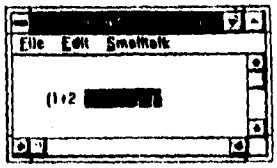


Figura 2. 35 Intento de evaluar "(1 + 2"

Esto no significa que tú debas cometer errores. Por lo contrario, es normal que cometas errores debes en cuando y que aprendas de ellos. Smalltalk/V es un ambiente que tolera errores y que te indica la forma de corregirlos. Esto se estudiará con mayor detalle en lecciones posteriores.

Descubriendo acerca de Instancias y Clases

Debemos ir más lejos, todos nuestros ejemplos han tratado con números como 1, 2, y 3. Siendo más precisos podemos llamarlos enteros. Afirmamos que 1 es un entero o decimos que 1 es una **instancia** de la **clase** enteros (integer). Además, tenemos el caso de que 1 es también una **instancia** de la **clase** número. Como 2 y 3 son instancias de las clases entero y número, queda claro que una clase tiene muchas instancias en general y que una instancia puede ser miembro de varias clases. Sólo para estar seguros que hemos entendido esta noción básica, revisemos una vez más:

....., -3, -2, -1, 0, 1, 2, 3, ... son enteros; instancias de la clase enteros.

....., -3, -2, -1, 0, 1, 2, 3, ... son también números; instancias de la clase número.

Hay muchos enteros (la clase enteros tiene muchas instancias)

La clase número tiene muchos números (la clase número tiene muchas instancias).

Si nos enfocamos sobre una clase particular tal como 1, nosotros podemos agregar -basados en la intuición y el sentido común- que un 1 es ambos, un entero y un número. Esto es algo fortuito si se usa Smalltalk/V en su terminología intuitiva. Para ser más precisos, podemos preguntar dentro del ambiente de Smalltalk/V que es un 1. ¿ A quién pertenece ?. Para hacer esto, evalúa lo siguiente:

1 class

obtendrás la respuesta **SmallInteger**

En esencia, nosotros preguntamos -"¿Cúal es tú clase? y Smalltalk/V respondió SmallInteger. La respuesta no fue un entero, como esperabamos tal vez. ¿Y qué más podemos saber sobre 1 y smallInteger ?, ¿hay algo superior a smallInteger ?. Intenta lo siguiente:

smallinteger superclass

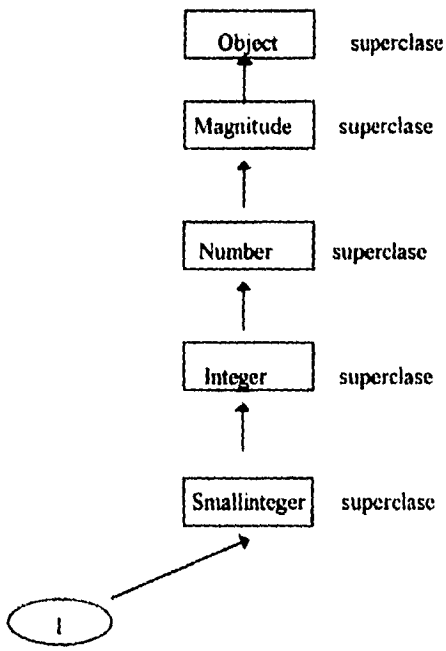
obtendrás la respuesta **Integer**

Aqui, nosostros preguntamos a la clase Smallinteger, "¿Cúal es tú superclase (que clase esta arriba o es superior a tí)?". La respuesta fue Integer. Así un SmallInteger es un Integer. ¿Nosotros podemos continuar esta línea de preguntas ?

Integer superclass	obtenemos Number
Number superclass	obtenemos Magnitude
Magnitude superclass	obtenemos object
Object superclass	obtenemos nil

¿Qué significa todo esto ?. Esto significa que 1 es simultáneamente un smallInteger, un entero, un número, una magnitud, y un objeto. Observa que nos detuvimos al llegar a un Objeto (este no tiene una supercalse). Esto lo podemos ilustrar mejor esquemáticamente, como en la figura 2.36. Nuestro entendimiento intuitivo fue correcto, ya que 1 es un entero y un número. Pero esto sólo fue una parte de la historia.

Figura 2. 36.- La instancia 1 y las clases que consideran a 1 como un miembro.



Desde luego, un 1 es un entero pequeño (samallInteger), y lógicamente existen enteros muy grandes (LargeIntegers). Consideremos las siguientes instancias de números grandes y evalúalos en alguna ventana de trabajo en Smalltalk/V.

123 class obtenemos Smallinteger

12345 class obtenemos Smallinteger

1234567890 class obtenemos LargePositiveInteger

Esto es significativo. En el último caso la respuesta no fue Largeinteger, como esperabamos. Esto sugiere que existen enteros pequeños negativos (small negative integers) y enteros grandes negativos (large negative integers) también.

Para concretar lo que estamos haciendo, evalúa las siguientes espresiones:

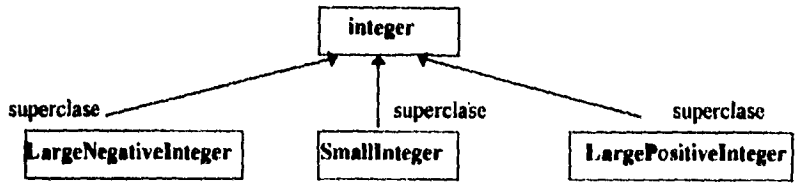
-1 class obtenemos Smallinteger

-123567890 class obtenemos LargeNegativeInteger

0 class obtenemos Smallinteger

Nota que para valores negativos pequeños (-1), no existe una clase especial ya que Smalltalk/V considera al -1 como miembro de la clase SmallInteger. Por otra parte, al evaluar un número negativo muy grande, si encontramos una clase específica para tales números. Y por último si preguntamos por la clase a la cual pertenece el cero (0), smalltalk/V responde como SmallInteger también, que significa que tampoco existe una clase especifica para el cero (0).
Toda esta información la podemos representar en un diagrama, como se ilustra en la siguiente figura:

Jerarquía de clases de integer



CAPITULO 3 Objetos, Mensajes y Estructuras de control

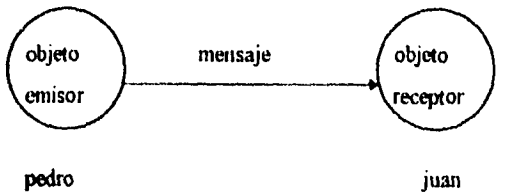
¿ Qué es un objeto ?

Las personas nos formamos conceptos desde temprana edad. Cada concepto es una idea particular o una comprensión de nuestro mundo. A estas cosas a las que se aplican nuestros conceptos se llaman objetos. Un objeto podría ser real o abstracto, como los siguientes:

- una factura
- una organización
- una pantalla con la que interactúa un usuario
- un mecanismo en un dispositivo de robótica
- todo un plano de ingeniería
- un componente de un plano de ingeniería

¿ Qué es un mensaje ?

Para Smalltalk/v todo es un objeto. Pero hablar de objetos es hablar de mensajes, ya que guardan una relación de funcionamiento mutúa. Observa la siguiente figura:



En la figura anterior un objeto emisor (pedro) envia un mensaje (hola) a un objeto receptor (juan). Esta situación se da de una forma natural. Asi funciona Smalltalk/v enviando mensajes entre objetos para crear situaciones especificas.

Todos los mensajes en Smalltalk/v tienen la misma sintaxis básica. El receptor del mensaje (el objeto que recibe el mensaje) aparece primero seguido por el mensaje. Como ejemplo envia un mensaje para abrir una ventana "inspector" sobre la clase **GraphicsDemo** :

GraphicsDemo inspect.

Nota lo siguiente, al receptor GraphicsDemo se le esta enviando el mensaje inspect, que le indica "inspeccionate a ti mismo".

Tipos de Mensajes.

Los objetos son los bloques más básicos del lenguaje Smalltalk/V. Ellos son análogos a segmentos de datos en otros lenguajes. Por ejemplo,

'esto es una cadena'

es un objeto en Smalltalk/V, una simple cadena de caracteres. Existen otros objetos en Smaltalk que tienen sus contrapartes en otros lenguajes:

1234 **"un entero"**

$A **"un simple caracter"**

#(1 2 3) **"un arreglo formado por enteros (objetos)"**

Observa el último ejemplo, el arreglo. Este es un objeto que contiene en si mismo otros objetos. Veamos otros ejemplos

#('array' 'de' 'cuatro' 'cadenas')

#('array' 'de' 5 'cadenas' 'y' 2 'enteros')

como podemos ver en los últimos ejemplos, los objetos contenidos dentro de otros objetos, no tienen que ser del mismo tipo o de igual magnitud. Parte del poder de Smalltalk se fundamenta en esta capacidad. Consideremos un objeto más complejo:

#(1 ('dos' 'tres') 4)

Este es un objeto (un arreglo) con tres objetos dentro de él. El segundo objeto en el arreglo es otro arreglo de dos cadenas.

Mensajes Simples

Un objeto no puede notificar algo por si mismo. En Smalltalk/V, se deben enviar mensajes a los objetos para que se realicen acciones. Los mensajes son similares a las llamadas de funciones en otros lenguajes. Por ejemplo mira la siguiente expresión en Smalltalk/V, compuesta de un mensaje simple:

20 factorial

Aqui se envia el mensaje **factorial** al objeto **20**. Para evaluar esta expresión, ve al tutorial y abre el capítulo 3 o teclea la expresión en una hoja de trabajo y selecciónalo. Entonces usa el **show it** del menú para compilarlo y evaluar el texto resaltado. El resultado debe ser un entero muy grande: **2432902008176640000**

Probemos con otro mensaje simple. Cuando seleccionamos y evaluamos la siguiente expresión, el resultado debe ser el entero 12, que indica la magnitud de la cadena:

'Llego la hora' size

Un mensaje esta compuesto de tres partes: un objeto receptor, un mensaje selector, y cero o más argumentos. En el ejemplo anterior, la cadena es el objeto receptor, el mensaje selector es el **size**, y no hay argumentos. Considera lo siguiente:

#(1 3 5 7) at: 2

en este ejemplo, el arreglo es el receptor, **at:** es el mensaje selector, y el **2** es el argumento.

Un mensaje siempre retorna un objeto particular como resultado. Se asemeja a las funciones en otros lenguajes. Similarmente, el mensaje selector es como el nombre de la función, y el objeto receptor es como el primer parámetro de la función. El ejemplo anterior pregunta pro el segundo elemento del arreglo, dando como resultado el valor de 3.

Ahora intenta evaluar este ejemplo:

'20' factorial

Una ventana **Walkback** aparece. Ya que 20 esta encerrado entre comillas, es una cadena, por lo que el factorial no tiene sentido. Como ilustra este ejemplo, los objetos saben que mensajes son apropiados para ellos. La enciclopedia de clases lista todos los tipos diferentes de objetos provistos en Smalltalk/V, y los mensajes a los que responden. Para libramos de la ventana **Walkback**, selecciona **close** en el sistema de menú de la ventana Walkback.

Mensajes Unarios

Los mensajes que carecen de argumentos, son llamados mensajes unarios. Como ejemplo evalúa los siguientes mensajes unarios:

#('arreglo' 'de' 'cadenas') size

'es tiempo de aprender' asUpperCase

'estoy bien' reversed

#(4 'cinco' 6 7) reversed

$A asciiValue

65 asCharacter

Mensajes Clave (Keyword)

Los mensajes con uno o más argumentos son llamados mensajes clave. Como ejemplo evalúa los siguientes mensajes clave:

'es tiempo de aprender' at: 6

'Hola' includes: $o

'Hola' at: 1 put: $H

'el pequeño gato' copyFrom: 3 to: 10

En los últimos dos ejemplos, el mensaje selector **at:put:** y **copyFrom:to** están separados por los argumentos. Cada argumento, esta precedido por dos puntos. En este ejemplo **at:put:** y **copyFrom:to:** trabajan con cadenas,pero estos mismos mensajes trabajan también con arreglos:

#(9 8 7 6 5) at: 3

#(1 (2 3) 4 5) includes: #(2 3)

#(1 0 4 5) at: 2 put: #(2 3)

#(9 8 7 6 5) copyFrom: 1 to: 2

En los ejemplos anteriores podemos notar otra caracteristica importante: diferentes tipos de objetos pueden responder a los mismos mensajes por medios diferentes. Los arreglos anteriores responden en forma distinta a los mensajes selectores **at:put:** y **copyFrom:to:** que las cadenas en los ejemplos previos.

Mensajes Aritméticos.

La aritmética en Smalltalk/V es similar a la de otros lenguajes. Por ejemplo:

3 + 4

pero, como otras expresiones en Smalltalk/V, esto es un mensaje. El entero 3 es el receptor, + es el mensaje selector, el entero 4 es el argumento, y el entero 7 es el resultado.

A continuación tenemos algunos mensajes aritméticos de los más comunes para evaluar:

5 * 7 **" multiplicación "**

5 // 2 **" división entera con truncación "**

4 \\ 3 **" calcula el resto "**

2 / 6 **" división relacional "**

La última expresión ilustra la aritmética relacional, o la aritmética de fracciones. En Smalltalk/V la aritmética relacional es exacta; no hay redondeo o truncamiento. Esto es porque Smalltalk/V representa el resultado en un objeto llamado fraction como un numerador y un denominador, más que calcular y aproximar a un valor numérico real.

Las exprosiones aritméticas en Smalltalk/V difieren con respecto a otros lenguajes en el orden de evaluación. En Smalltalk/V las expresiones aritméticas se evalúan estrictamente de izquierda a derecha, con ninguna precedencia entre los ordenadores. Por ejemplo, evalúa la siguiente expresión:

```
3 + 4 * 2
```

El resultado es 14, no 11. Pero, es posible usar paréntesis para controlar el orden de evaluación, como en el siguiente ejemplo:

```
3 + 4 * 2
```

También se puede usar la aritmética de punto flotante, como sigue :

```
1.5 + 6.3e2
```

Mensajes Binarios

Los mensajes aritméticos son ejemplos de mensajes binarios, mensajes con un argumento y uno o dos caracteres especiales (o digitos y letras)como el selector. Los mensajes binarios son siempre evaluados estrictamente de izquierda a derecha, a menos que se utilicen paréntesis. Por ejemplo evalúa estos mensajes que no son aritméticos:

```
'hola', 'todos'
#(1 2 3), #(4 5 6)
```

En estos ejemplos, el carácter especial es la coma. Esta concatena los argumentos con el objeto receptor.

Mensajes dentro de Mensajes

Como hemos visto antes, los mensajes son como funciones que retornan un objeto. cuando un objeto aparece en una expresión, podemos enviarle un mensaje que retorne un objeto de tipo similar. Por ejemplo evalúa cada uno de los siguientes:

'hola' size + 4

'ahora' size + #(1 2 3 4) size

#(1 12 24 36) includes: 4 factorial

En la última expresión anterior hay realmente dos mensajes. **Factorial** es un mensaje unario, y es procesado primero. El resultado llega a ser el argumento para el mensaje **includes**. Ahora evalúa una expresión más compleja (no son necesarios paréntesis):

4 factorial between: 3 + 4 and: 'hola' size * 7

Esta expresión esta compuesta por cinco mensajes. Los cinco mensajes selectores son **factorial**, **size**, +, *, y **between:and:**. Como puedes ver los mensajes unarios son evaluados primero que los mensajes binarios, que en turno son evaluados antes que los mensajes clave(keyword). Como es usual, podemos alterar esta precedencia utilizando paréntesis:

'hola' at: (#(5 3 1) at: 2)

La expresión esta compuesta de dos mensajes **at:**, uno para el arreglo y otro para la cadena. Observa que pasa cuando no usamos paréntesis, pruébalo:

'hola' at: #(5 3 1) at: 2

Esta expresión es un mensaje simple con dos argumentos. El mensaje selector es **at: at:**, algo que no teníamos en mente.

Expresiones en serie.

No es posible hacer grandes cosas con tan solo expresiones simples. A continuación se dan una serie de expresiones que puedes evaluar como una unidad particular. Selecciona la serie completa y evalúa con **Do It** en el menú de smalltalk/V:

Window turtleWindow: 'Turtle Graphics'.

Turtle black.

Turtle home.

Turtle go: 100.

Turtle turn: 120.

Turtle go: 100.

Turtle turn: 120.

Turtle go: 100.

Turtle turn: 120

Cada mensaje en la serie esta separado de el próximo por un punto. Se colocan de esta manera solo para dar mejor apariencia. La primera línea abre una ventana para dibujar y crea el objeto **Turtle**, una pluma. El receptor de todos los mensajes en las líneas siguientes es el objeto **Turtle**. Para conseguir una pintura diferente cierra la ventana gráfica turtle y cambia la palabra **ClrBlack** por **ClrDarkGray** en la expresión en serie y evalúa nuevamente.

Mensajes en cascada

Los mensajes en cascada son un camino corto para escribir series de mensajes que son enviados al mismo receptor. Por ejemplo, la expresión siguiente dibuja la misma figura que el ejemplo previo (solo usa una línea y un color diferente):

```
Window turtleWindow: 'Turtle Graphics'.
Turtle
  defaultNib: 2;
  foreColor: ClrDarkgray;
  home;
  go: 100;
  turn: 120;
  go: 100;
  turn: 120;
  go: 100;
  turn: 120
```

El receptor turtle esta escrito únicamente una vez, y cada mensaje (excepto para el último) esta terminado con un punto y coma en vez de un punto. La identación nuevamente, es opcional; esto hace simplemente el código más legible.

Loops simples

Nosotros podemos simplificar el ejemplo anterior usando un mensaje que repita un loop cierto número de veces:

```
Window turtleWindow: 'Turtle Graphics'.
  Turtle
    black;
    home.
  3 timesRepeat: [ Turtle go: 100; turn: 120 ]
```

En este ejemplo, el argumento para el mensaje **timesRepeat:** es un bloque de código. Los bloques de código están escritos como una serie de mensajes encerrados en paréntesis cuadrados, [como estos]. Nosotros escribimos el ejemplo de arriba generalmente como :

```
Window turtleWindow: 'Turtle Graphics'.
Turtle
  black;
  home.
3 timesRepeat: [
  Turtle
    go: 100;
    turn: 120]
```

Esto hace que el mensaje en cascada sea más fácil de leer dentro del bloque. Cierra el Turtle Graphics antes de continuar.

Variables Temporales

Se llaman variables temporales porque Smalltalk/V las desecha una vez que ha hecho uso de ellas. Estas variables son declaradas encerradas entre barras verticales en la primera línea de las expresiones en series. Los nombres de estas variables deben empezar con una letra minúscula, mientras que el resto del nombre puede ser alguna combinación de minúsculas y mayúsculas y dígitos. Por ejemplo a continuación tenemos un breve programa que usa tres variables temporales y un loop para procesar un arreglo de varios factoriales:

```
| cuenta indice factoriales |
factoriales := #( 3 4 5 6 ).
indice := 1.
factoriales size timesRepeat: [
  cuenta := factoriales at: indice.
  factoriales at: indice put: cuenta factorial.
  indice := indice + 1].
^factoriales
```

En la primera linea se declaran tres variables temporales: cuenta, indice, y factoriales. Una variable temporal puede sostener algún tipo de objeto. Para darle a esta un valor, se debe usar una expresión de asignación.

Expresiones de asignación.

En los ejemplos anteriores se usaron cuatro expresiones de asignación:

```
factoriales := #(3 4 5 6).
indice :=1.
cuenta :=factoriales at: indice.
indice :=indice + 1.
```

Las primeras dos asignan objetos a las variables temporales, mientras que las últimas dos asignan el resultado de mensajes. como el resultado de un mensaje es siempre un objeto particular, ellas actúan asignando objetos a las variables temporales.

Expresiones de Retorno

En el ejemplo anterior del factorial la última expresión es:

^factoriales

El signo "^" significa que este es un valor que debe ser retornado como el resultado de una expresión en serie. De tal forma que la declaración que inicia con un signo "^" es llamada expresión de retorno.

Variables Globales

Smalltalk/V tiene muchos objetos construidos , muchos de los cuales se encuentran dentro de variables globales. Se llaman variables globales porque Smalltalk/V no las destruye cuando termina el uso de las mismas. Por ejemplo, Smalltalk/V proporciona (entre otras) las siguientes tres variables globales:

Display
Transcript
Disk

Display es una instancia de la clase **Screen** y es un objeto que mantiene toda la información

sobre lo que tú veas en tu monitor. Las variables globales siempre contienen un objeto particular. Además un objeto puede tener variables de instancia que contengan otros objetos. Por ejemplo selecciona **Display** , y usa el **Show It** para ver el contenido del estado actual. El resultado es en si mismo un objeto particular.

Los nombres de las variables globales siempre empiezan con una letra mayúscula y el resto con mayúsculas y minúsculas o dígitos. Escribe lo siguiente, selecciónalo, y haz un **Show It**:

Daniel

Cuando la variable global no existe, aparecerá una caja de diálogo donde podrás elegir si deseas crear la variable o no. Si no la creas smalltalk/V asume que haz cometido un error, y muestra un mensaje de error. haz click en el botón No para liberar la variable **Daniel** indefinida. Esto te protege de crear variables globales cuando tecleaste algo erróneo. Igual que en las variables temporales, se deben usar sentencias de asignación para asignar valores a las variables globales:

Daniel := ' Daniel Martinez'

Evalúa con **show it** y da click en el botón **Yes** para definir esto como una nueva variable global.

Estructuras de Control

En temas anteriores hemos aprendido algo de las expresiones básicas que maneja Smalltalk/V. Pero como en otros lenguajes, Smalltalk/V no puede hacer mucho si no tiene la capacidad de evaluar una condición, y realizar una acción basado sobre el resultado obtenido, o también repetir una acción varias veces de acuerdo con un programa específico. En este apartado aprenderemos las expresiones condicionales y las estructuras de control que existen en Smalltalk/V.

Te recordamos que estos ejemplos se encuentran en el capitulo 3 del tutorial. Solo necesitas abrir el archivo y evaluar los ejemplos.

Comparación de Objetos

Smalltalk/V compara objetos enviando mensajes. Realiza las comparaciones clásicas de "<", "<=", ">", ">=", "=" que son implementados como mensajes binarios. Como ejemplo, evalúa cada una de las siguientes expresiones :

3 < 4

#(1 2 3 4) = #(1 2 3 4)

Todos los objetos entienden la igualdad. "=". Muchos objetos definen operadores relacionales, o el orden de mensajes, como en el siguiente ejemplo:

```
'hola' <= '¿Cómo estas?'
```

Si en la comparación hay mensajes binarios, a menudo se usan paréntesis si hay otro mensaje binario en la expresión. Por ejemplo evalúa las siguientes expresiones con y sin paréntesis:

```
5 = (2 + 3)
5 = 2 + 3
```

Como puedes ver al evaluar estos ejemplos, las comparaciones retornan falso o verdadero.

Prueba de Objetos.

Muchos objetos entienden los mensajes que se les envían, permitiendo hacer pruebas sobre su estado o condición. Por ejemplo evalúa cada una de las siguientes expresiones:

```
$a isUpperCase
('hola' at: 1) isVowel
7 odd
```

Estos mensajes retornan verdadero o falso también.

Ejecución Condicional.

Muchos lenguajes utilizan la declaración **if** para condicionar la ejecución de una serie de declaraciones. Smalltalk/V usa bloques de código y mensajes para realizar la misma acción. Por ejemplo, considera la siguiente expresión, que calcula el mayor de dos números:

```
| max a b |
a := 5 squared.
b := 4 factorial.
a < b
ifTrue: [max := b]
ifFalse: [max := a].
^max
```

El mensaje de comparación **a < b** retorna falso o verdadero, que entonces se convierte en el receptor del mensaje **ifTrue:ifFalse:** esto en turno, ejecuta el bloque correspondiente de código.

Como puedes ver, todos los objetos regresan un resultado. Evalúa la siguiente expresión:

```
3 < 4
ifTrue: ['la expresión es verdadera']
ifFalse: ['la expresión es falsa']
```

El mensaje **ifTrue:ifFalse:** regresa el resultado de la última expresión en el bloque que este ejecuta. Veamos otro ejemplo:

```
| cadena indice c |
cadena := 'Es tiempo de aprender'.
indice := 1.
cadena size timesRepeat: |
c := cadena at: indice.
cadena
   at: indice
   put:
      (c isVowel
         ifTrue: | c asUpperCase |
         ifFalse: | c asLowerCase | ).
      indice := indice + 1 |.
^cadena
```

El ejemplo anterior convierte todas las constantes minúsculas a mayúsculas, usando la declaración **c isVowel** para encontrar cuando se da esta situación y realizar la conversión.
Los otros mensajes que ejecutan un bloque de código condicional son **ifTrue:**, **ifFalse:**, **ifTrue:ifFalse:**, y **ifFalse:ifTrue:**.

Expresiones Booleanas

Hasta ahora hemos visto ejemplos que dependen de una simple comparación o de una prueba de objetos. Pero a menudo es necesario realizar una prueba o una acción más compleja. Para hacerlo es necesario utilizar los mensajes **and** y **or**. Por ejemplo, considera el siguiente fragmento de código, que prueba si un caracter es o no un digito:

```
( c < $0 or: [c > 9 ] )
```

El receptor del mensaje or es el resultado de la primera comparación, como falso o verdadero. El argumento es un bloque de código donde la última expresión retorna falso o verdadero también. El **and** trabaja de forma similar:

```
( c < $0  or: [c > 9 ] )
```

Para ver como se usan en un ejemplo completo, evalúa la siguiente expresión:

```
"Se calcula el primer entero en una cadena"
| cadena indice respuesta c |
cadena := '1234 es el número'.
respuesta := 0.
indice := 1.
cadena size timesRepeat: [
c := cadena at: indice.
(c < $0 or: [ c > $9 ] )
ifTrue: [^respuesta].
respuesta := respuesta * 10
+ c asciiValue - $0 asciiValue.
indice := indice + 1].
^respuesta
```

Nota la expresión de retorno **ifTrue:[^answer]** en la parte media del código. Esta expresión existe tan pronto como se encuentra un caracter que no es un digito.

Tú puedes hacer pruebas más complejas utilizando expresiones. Por ejemplo, observa el siguiente fragmento de código, que prueba si un caracter es digito o no en el intervalo de la A a la F :

```
( c isDigit  or: [c >= $A and : [ c< = $F ] ] )
```

Loops en mensajes.

Ya hemos visto algo sobre un loop simple, el timesRepeat:. A continuación damos una expresión que usa un loop simple para copiar un archivo:

```
"copiado de un archivo"
| entrada salida |
entrada := File pathName: 'tutor\capitulo.3'.
salida := File pathName: 'tutor\copycapi.3'.
[entrada atEnd]
    whileFalse: [salida nextPut: entrada next].
entrada close.
salida close
```

Nota el mensaje atEnd en el ejemplo anterior. Este mensaje retoma verdadero cuando no hay mas caracteres para ser leidos del archivo de entrada; de otra forma regresa falso. El archivo de entrada es leído con el mensaje next, que retoma el próximo carácter en el archivo. El archivo de salida es escrito con el mensaje nextPut, que escribe el argumento a la salida.

Iteradores generalizados

Los bloques con argumentos permiten a smalltalk/V suministrar mensajes llamados iteradores generalizados. Estos son : **do**, **select**, **rellect**, y **collect**.

El iterador "do".

Es el más sencillo de todos. En el siguiente ejemplo, se usa el iterador **do** para contar el número de vocales en una cadena.

```
"conteo de vocales en una cadena"
| vocales |
vocales := 0.
'Ahora es el tiempo' do: [ :char |
 char isVowel
      ifTrue: [ vocales := vocales + 1 ] ].
^vocales
```

Como puedes observar, el mensaje **do** itera a través de toda la cadena caracter por caracter.
El siguiente ejemplo es similar al anterior:

```
"conteo de vocales en una cadena"
| vocales cadena indice |
vocales := 0.
indice := 1.
cadena := 'Es hora de aprender Smalltalk/V'.
[indice <= cadena size]
  whileTrue: [
    (cadena at: indice) isVowel
      ifTrue: [vocales := vocales + 1].
    indice := indice + 1].
^vocales
```

El mensaje do puede también iterar arreglos. A continuación se da un ejemplo donde el do calcula la suma de un arreglo de enteros.

```
| dato suma |
dato := #( 1 2 3 4 5 6 7 8 9 10).
suma := 0.
dato do: [:elemento | suma := suma + element
^suma
```

La estrategia básica es acumular la respuesta en una variable llamada suma. Primero suma se inicializa a cero. Después son obtenidos en dato todos los elementos uno a uno usando el do. Cada vez que un elemento es obtenido la suma se actualiza para incluir el nuevo elemento usando la expresión suma := suma + elemento

En el siguiente ejemplo se usa nuevamente el do para calcular el producto de los elementos contenidos en un arreglo:

```
| dato producto |
dato := #( 1 2 3 4 5 6 7 8 9 10).
suma := 0.
dato do: [:elemento | producto := producto * elemento
^producto
```

El iterador **select**

Este es un iterador más poderoso, de mayor capacidad. Observa el ejemplo, donde se usa el mensaje **select** para contar el número de vocales en una cadena:

```
('llego la hora final' select: [ :c | c isVowel ]
  size
```

El mensaje select itera toda la cadena, caracter por caracter, y donde un caracter es vocal valida como verdadero. En este caso regresa como resultado la cadena de vocales obtenida a partir de la cadena original. El mensaje **size** nos dice cuantos elementos fueron seleccionados.

El iterador **reject.**

Es otro iterador generalizado, y trabaja de forma particular. En el siguiente ejemplo se calculan todos los digitos de los cuales su factorial sea menor que el mismo digito elevado a la cuarta potencia:

```
"Calcula todos los digitos cuyos factoriales sean menores que esos digitos
  elevados a la cuarta potencia"
#( 1 2 3 4 5 6 7 8 9 ) reject: [ :i |
  i factorial >= ( i * i * i * i ) ]
```

Este mensaje trabaja en forma similar al select; pero retorna todos los elementos del receptor para que el bloque de código retorne falso, además de verdadero.

El iterador **collect.**

Este mensaje evalúa el bloque de código para cada elemento del receptor y pregunta por todos aquellos valores que serán retornados. Evalúa el siguiente ejemplo:

```
"Cálculo del cuadrado de cada elemento en el arreglo"
  #(1 13 7 10) collect: [ :i | i * i ]
```

Para ayudarte a ver las diferencias entre el **select:**, **reject:**, y **collect:**, evalúa cada una de las siguientes expresiones:

```
#(1 2 3 4 5 6 7) select: [ :c | c odd ]
#(1 2 3 4 5 6 7) reject: [ :c | c odd ]
#(1 2 3 4 5 6 7) collect: [ :c | c odd ]
```

CAPITULO 4 Clases, Métodos y Herencia

Definición de una Clase

La solución de problemas usando Smalltalk/V involucra la clasificación de objetos de acuerdo a sus similitudes y sus diferencias. Hasta este momento has visto el comportamiento externo de los objetos mediante el envio de mensajes hacia ellos y observado sus resultados. Pues bien, una clase define el comportamiento de objetos similares mediante la especificación de sus interiores -es decir, las variables que ellos pueden contener y los métodos disponibles para responder a los mensajes que se les envíen.

Cada objeto es una instancia (miembro) de una clase. Por ejemplo, **#(1 2 3)** y **#(sam joe)** son instancias de la clase **Array**, del mismo modo tanto **'north'** y **'south'**, son instancias de la clase String. Todos los objetos saben a que clase pertenecen. Por ejemplo, evalúa las siguientes expresiones:

```
#(Francesca Jackie Mariaa Bree) class
'Rakesh Vijay Carles Robin Tyler' class
Turtle class
```

Las variables internas de un objeto son llamadas variables instancia, ellas, en sí mismas pueden contener otros objetos. Por ejemplo, los objetos en la clase **Fraction** tienen como variables instancia a **numerator** y **denominator**. Así en la representación de el objeto fracción 1/7, la variable instancia **numerator** contiene el objeto 1, y la variable instancia **denominator** contiene el objeto 7.

El Browser de Jerarquía de Clases.

Para programar en Smalltalk/V, utilizarás una ventana especial llamada Class Hierarchy Browser. Esta ventana te permite mezclar y cambiar las clases existentes y asi como las definiciones de los métodos, o también crear nuevas. Para comenzar a familiarizarte con esta ventana ábrela mediante la evaluación de la siguiente expresión:

```
ClassHierarchyBrowser new openOn: (Array with: Integer
with Fraction with: String with: GraphicsDemo)
```

Dicha ventana aparecerá en la pantalla. Tu puedes reajustar su tamaño o moverla como cualquier otra ventana de Smalltalk/V. Una ventana Class Hierarchy Browser esta ahora disponible para las clases Integer, Fraction, String y GraphicsDemo, como tú puedes ver esta ventana contiene algunas divisiones; en el recuadro superior izquierdo se encuentran las cuatro clases mencionadas. Selecciona el renglón donde se encuentra la clase **Fraction:** y observarás la siguiente ventana:

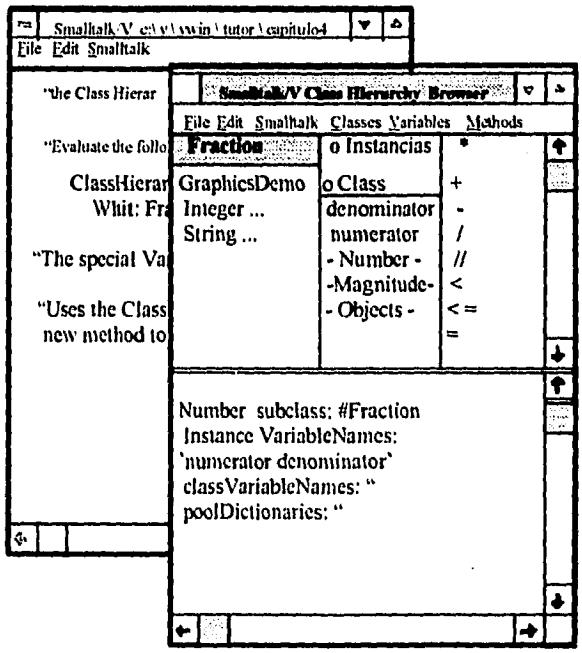


Figura 4.1 Ventana Class Hierarchy Browser

El recuadro superior derecho muestra los métodos definidos para la clase **Fraction**, que es la clase seleccionada. El recuadro del centro, entre las recuadros de las clases y de los métodos, hay una lista de las variables definidas en la clase y también las heredadas por la misma clase. El recuadro inferior muestra el mensaje de definición de la clase para la clase **Fraction**. El mensaje de definición de la clase muestra las características de comportamiento que contiene ésta. Nota que el argumento **instanceVariableNames:** ; es un string que especifica los nombres de las variables instancia, las cuales para este caso son **numerator** y **denominator** que también son desplegados como elementos en la lista

de el recuadro de Variables. Aprenderás acerca de otros argumentos que se pueden establecer dentro de el mensaje de definición de clase más tarde en el transcurso de este tutor.
Selecciona el método * en el recuadro superior derecho; el código fuente para el método aparece en el recuadro de abajo. Este recuadro es un editor de texto el cual tu puedes usar para cambiar los métodos existentes y crear nuevos. Prueba seleccionando otros métodos, y observa su código fuente.

Creación de una Clase

Cuando se crea una nueva Clase, debemos primero decidir que superclase será la que contendrá a la nueva, por ejemplo, 23 es un pequeño entero cuya superclase es la clase **Integer**. **Integer** por su parte, tiene como superclase la clase llamada **Number**, la que a su vez tiene como superclase a la clase **Magnitud**, la que por último tiene como superclase la clase **Object**. Sin embargo, si se tiene duda de cual debe ser la superclase de la nueva que quieres crear, es bueno probar eligiendo como superclase a la clase **Object**.

Los pasos básicos para crear una clase son los siguientes:

- Selecciona la clase que será la superclase de la tuya.
- Selecciona en el menú **Classes** (todo esto dentro del Class Hierarchy Browser) la opción **Add Subclass**, una ventana de diálogo aparecerá en la pantalla.
- En esta ventana deberás aportar el nombre de la nueva clase que quieres crear, especificar si contendrá variables indexadas o no, y si estas variables contendrán apuntadores (pointers) o bytes.

Cuando presiones el botón **Ok** aparecerá la definición de tu nueva clase sin **instanceVariableNames**, **classVariableNames** ni **poolDictionaries**; los que deberás agregar ahora o durante el transcurso de la vida de la misma clase.

Variables de Instancias.

Los objetos pueden contener tanto variables de instancia nombradas o indexadas. Las variables de instancia nombradas son accesadas mediante un nombre, como con **numerator** para los objetos **Fraction**. Las variables de instancia indexadas son identificadas mediante enteros comenzando por 1. Ellas son siempre accesadas por mensajes como:

'lugar' at: 2

'partes' at: 5 put: $y

Una clase de objeto especifica las variables instancia nombradas y si variables instancia indexadas pueden ser usadas en sus instancias. El número de variables instancia nombradas es establecido por todas las instancias de la clase. El número de variables instancia indexadas es definido cuando creas el objeto y puede diferir entre instancias de una clase. Por ejemplo, los dos strings de arriba tienen cinco y seis variables indexadas, respectivamente.

Eliminación de una clase.

Eliminar una clase es más simple que crearla. Los pasos básicos son los siguientes:

- Selecciona la clase que será eliminada.
- Selecciona en el menú **Classes** la opción **Remove Class**; a continuación una ventana de confirmación aparecerá.
- Haz click sobre el botón **Yes** o presiona la tecla Enter, a menos que hayas cambiado de opinión, en cuyo caso presiona el botón **No**.

Localización de una clase

Una forma de localizar una clase es desplegando las clases en el recuadro correspondiente dentro de el **Browser**, abriendo y cerrando secciones hasta que encontremos la clase que deseamos. Esto puede tomar algo de tiempo si no sabemos donde está localizada la clase. Por ejemplo, supón que deseamos encontrar la clase **Pen**. Después de buscar sin éxito por algún tiempo, seguramente trataremos de encontrar otro camino más práctico. Sin embargo Smalltalk/V provee de una forma más rápida, que es como sigue:

- Selecciona en el menú **Classes** la opción **Find Class...** .
- Un prompter aparecerá, este prompter preguntará por el nombre de la clase a ser localizada. El nombre de la clase debe ser tecleado correctamente; de otro modo, esta no será encontrada. Recuerda que el nombre siempre comienza con una letra mayúscula.
- Una vez que el nombre de la clase es tecleado, debes presionar el botón **Ok** o presionar la tecla Enter para iniciar la búsqueda.

Definición de un Método.

Los métodos son código Smalltalk/V, son los algoritmos que determinan el comportamiento de un objeto y su performance. Se pueden comparar a definiciones de funciones en otros lenguajes. Cuando un mensaje es enviado a un objeto, un método es evaluado, y un objeto a su vez es retornado como resultado. Por ejemplo, evalúa la siguiente expresión:

```
(1/7) numerator
```

Cuando el mensaje **numerator** es enviado a la fracción 1/7, Smalltalk evalúa el método **numerator** definido en la clase **Fraction:**

```
numerator
        ^numerator
```

La primera línea de el método define su nombre. (Es importante notar que este corresponde a el selector en el mensaje mostrado). La segunda línea retoma el resultado **numerator**, la variable instancia de el objeto receptor fracción. Como un ejemplo más complejo se presenta el siguiente, evalúa el siguiente mensaje:

```
(2/3) * (5/7)
```

Enviando el mensaje * a la fracción 2/3 con la fracción 5/7 como argumento se evalúa el método * en la clase **Fraction**:

```
* aNumber
        ^(numerator * aNumber numerator)/
                (denominator * aNumber denominator)
```

La primera línea define el nombre de el método (*) y el nombre para el argumento, aNumber, el cual es usado en el resto de el método para representar el objeto argumento. el método retoma una nueva fracción en la cual el numerador es el producto de los numeradores receptor y argumento, y el denominador es el producto de los denominadores receptor y argumento.

Nota que **numerator** y **denominator** aparecen ambos como variables instancia y mensajes en este método. En este ejemplo, el argumento **aNumber** contiene la fracción 5/7, mientras las variables instancias **numerator** y **denominator** contienen a los números 2 y 3 respectivamente.

Como puedes ver de este ejemplo, los objetos de Smalltalk/V son tipos de datos abstractos. El operador multiplicación aparece sobre la mitad de el objeto receptor (2/3), cuyas variables internas numerator y denominator son accesibles. El argumento es otro objeto (5/7). Aunque este es de la misma clase de el receptor, sus variables internas no son disponibles en este método, y por lo tanto los mensajes pueden ser usados para responder la información deseada. Esta característica de Smalltalk/V provee completa seguridad para manipulación exterior.

Creación de un método

Para agregar un método a cualquier clase ya existente, se deben seguir generalmente los siguientes pasos:

- En el Class Hierarchy Browser, la clase que será editada. La definición de la clase aparecerá en el vidrio de texto.
- Asegúrate que el switch de instancia este seleccionado si un método instancia es el que se va a construir. Por ejemplo, para un método clase, se selecciona el switch classs.
- Selecciona en el menú **Methods** la opción **New Method**. Un bosquejo de el formato de un método entonces aparece en el vidrio de texto.
- Edita el bosquejo del método, es decir, reempleza "message Pattern" por el nombre de tu nuevo método, elimina las demás líneas para que ahí puedas poner tus líneas de código que darán forma a tu método.
- Selecciona de el menú **File** la opción **Save**. Ese nuevo método entonces aparecerá en la lista de el vidrio de lista de métodos para esa clase.

La variable especial "self ".

Ahora añadiremos el siguiente nuevo método a la clase Fraction:

```
fraction
"Responde el receptor menos su parte entera."
^self - self truncated
```

Este método retorna una fracción menor de uno: el receptor de el mensaje menos su parte entera de el receptor. El método contiene la palabra self, una variable especial que representa el objeto el cual es el receptor de el mensaje fraction. Adiciona el método a la clase **Fraction** usando los siguientes pasos:

- Activa el menú **Methods** y selecciona **New Method**.
- Verás un "machote" de un método en el recuadro inferior. Reemplázalo con el código fuente para el método fraction definido arriba.
- Activa el menú **File** y selecciona **Save** o presiona las teclas Alt + S.

Smalltalk/V compila el nuevo método y lo instala en la clase **Fraction**. Pruébalo evaluando los siguientes mensajes:

```
(22/7) fraction
(2/3) fraction
```

Agregando un Método a el programa de Gráficos.

En el capitulo 3 usaste una serie de expresiones para dibujar una flor de poligonos en el ventana Turtle Graphics. El siguiente ejemplo empaqueta estas expresiones dentro de un método, y extiende el programa de Demo Graphics para que puedas escoger dibujar una flor de poligonos en el menú del demo de el programa de Gráficos.

Para crear el nuevo método, agrega el siguiente método a la clase GraphicsDemo usando el Class Hierarchy Browser:

```
polyFlower
"Dibuja una flor de polígonos de tamaño especificado por el usuario."
```

```
        | tamañoFlor longitud |
        tamañoFlor := Prompter
        prompt: 'Número de lados?'
        defaultExpression: '30'.
    graphs removeKey: #polyFlower ifAbsent: [ ].
    self
      drawBlockNow: |
      longitud := 240 // tamañoFlor.
      pen
        erase;
        home;
        north.
      tamañoFlor timesRepeat: |
        pen
          up;
          go: longitud // 2;
          down;
          go: longitud.
      tamañoFlor - 1 timesRepeat: |
        pen
          turn: 360 // tamañoFlor;
          go: longitud] | |
  for: #polyFlower
```

Este método difiere de las expresiones de flor de polígonos usadas en los primeros capítulos en dos maneras. Primeramente, el método **polyFlower** usa un prompter de diálogo, en vez de una constante, para determinar el número de lados a dibujar. En segundo lugar, el método **polyFlower** usa la variable instancia **pen**, en vez de la variable global **Turtle**, para dibujar la flor de polígonos. Nota también que el código familiar de turtle-graphic aparece en un bloque pasado a **drawBlockNow:for:** en el método **polyFlower**. Este método implementa el modo de dibujo segmentado en el cual las gráficas son capturadas en un formato de "repetición instantánea".

Para hacer este nuevo dibujo disponible desde dentro de el Demo de Graphics, necesitas modificar las selecciones de el menú Graphics. Para agregar polyFlower como un opción en el menú del demo de

Graphics, edita el método **graphicsMenu:** en la clase **GraphicsDemo** como sigue (no olvides ajustar el selector **lines:** a #(6) o la línea de separación aparecerá en un lugar incorrecto):

```
graphicsMenu: aPane
    aPane setMenu: ( (Menu  labels: 'Poly Flower\&Walk Line\&Mandala\ &Multi
Mandala\&Dragon\Multi &Spiral\&Paste' withCrs
lines: #(6)
selectors: #(polyFlower walkLine mandala multiMandala dragon
multiSpiral paste ) )
title: '~Graphics';
    owner: self;
    yourself)
```

Ahora prueba el programa demo seleccionando **Graphics Demo** de el menú **File** de la ventana **Transcript.** Encontrarás **polyFlower** como la nueva primera selección en el menú **Graphics** de la ventana de Graphics Demo. Nota que tu flor se redibuja a si misma automáticamente cuando cubres y descubres la ventana de Graphics Demo con la ventana de **Transcript**.

Eliminación de un Método

Para eliminar un método tan sólo se deben seguir los siguientes pasos:

- En el recuadro de la lista de métodos, selecciona el método que va a ser eliminado.
- Selecciona del menú **Methods** la opción **Remove**. Aquí cabe hacer un aviso importante, para este caso no existe una ventana de confirmación de la eliminación; por lo que si se comete un error no hay manera de recuperarlo.

Variables de las Clases

Las variables de clase son variables globales accesibles a todas las instancias de una clase. Ellas son usadas para compartir datos dentro de una clase. Las variables de clase comienzan una letra mayúscula.

Agreguemos una variable de clase a GraphicsDemo para contar el número de veces que ejecutamos el

método **mandala** del demo de **Graphics**. Primero, define la nueva variable de clase usando el Class Hierarchy Browser. Selecciona **GraphicsDemo**, ahora edita la definición de clase para tener el nombre **MandalaCount** en el argumento de **classVariableNames:** . Entonces selecciona **Save** en el menú **File** o presiona las teclas Alt + S. Esto crea la variable de clase y recompila la clase **GraphicsDemo**. Ahora agrega el siguiente método instancia a **GraphicsDemo**:

```
mandalaCount
        ^MandalaCount
```

Entonces agrega el siguiente código a el final de el método mandala en GraphicsDemo:

```
    MandalaCount isNil
ifTrue: [MandalaCount := 1]
ifFalse: [MandalaCount := MandalaCount + 1]
```

Recuerda poner un punto después de **#mandala** al final de el método **mandala** antes de agregar las sentencias de arriba. Esto es necesario para mantener la sintaxis propia de Smalltalk/V.
Para ver cuantas veces mandala ha sido dibujada, evalúa la siguiente expresión antes y después de correr al programa demo:

```
GraphicsDemo new mandalaCount
```

Modificación de Métodos

Cuando un Método ya existe pero deseamos cambiar alguna de sus características de comportamiento o modificarlo en algún sentido estos son los pasos esenciales a seguir:

- Selecciona ya sea el switch de instancia o clase para tener acceso a el tipo de método que quieres modificar de la clase deseada.
- Selecciona el método de el vidrio que contiene los métodos.
- Edita el método en el vidrio de texto, es decir, realiza las modificaciones necesarias para que el método tenga el nuevo compartamiento deseado.
- Selecciona el menú **File** y elige la opción **Save**. El método se salvará con el nuevo nombre y con los cambios realizados. En caso de que hayas cambiado el nombre, el anterior desaparecerá y en su lugar estará el nuevo.

Herencia y Jerarquía de clases

Mucho de el poder de Smalltalk/V viene de el arreglo de sus clases en una jerarquía. Cada clase tiene una superclase inmediata y posiblemente uno o más subclases, con la clase Object en la parte más alta de la jerarquía. Tu ya estas familiarizado con este mismo sistema en biología, el cual arregla los organismos vivientes en clases, basados sobre características comunes a cada clase. Las clases más altas en la jerarquía representan características más generales, mientras las clases más bajas en la jerarquía representan características más particulares. Por ejemplo, pez y árbol son más abstractos que ballena y olmo.

Ya hemos visto como Smalltalk/V organiza su código (métodos) mediante las clases. En este y en los siguientes capitulos, verás como puedes desarrollar soluciones de problemas genéricos usando clases abstractas, y entonces desarrollar más soluciones de aplicación especifica las cuales "especializan" la solución general por la adición de una pequeña suma de código en las subclases.

Si todavia no cierras la ventana Class Hierarchy Browser abierta en los ejemplos anteriores, ciérrala en este momento. Dentro de el menú **File** en la ventana **Transcript**, selecciona **Browse Classes**. Selecciona la clase **Boolean** en la lista de el recuadro de clases. Escoge **Show Sublasses** de el menú **Classes**. Ahora selecciona la clase **True** la cual te muestra la siguiente ventana:

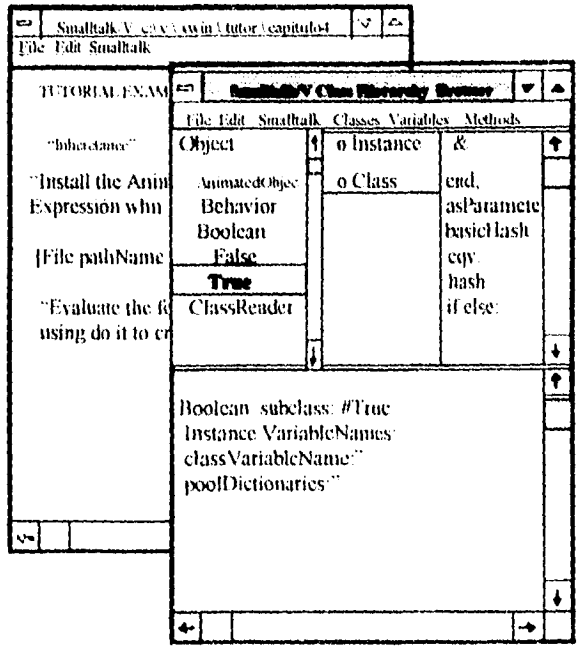


Figura 4. 2 La clase True

Nota que las clases en la lista de el recuadro de clases existen sangrías. Las sangrías muestran la jerarquía de clases. Cada clase es una superclase de las clases con sangría bajo ella, como puedes ver, **Object** es la superclase de todas las clases, y **Boolean** es la superclase de **True** y **False**.

Una clase con "..." seguidos a su nombre tiene subclases que no están siendo desplegadas. Cuando abres el Class Hierarchy Browser, este despliega sólo el primer nivel de subclases de la clase **Object**. Para desplegar u ocultar las subclases de una clase, usa **Show Subclasses** o **Hide Subclasses**, respectivamente en el menú **Classes** o haz doble click con el Mouse sobre el nombre de la clase. Prueba ocultando las subclases de la clase **Boolean** usando la técnica de el doble click.

Herencia de las variables instancia.

Un objeto hereda todas las variables de instancia definidas en su superclase además de las contenidas en su propia clase. Por ejemplo, los pericos, pingüinos, perros y ballenas cada uno podrían contener las siguientes variables instancia:

Parrot

(name, knowledge, habitat, topSpeed, color, vocabulary)

Penguin

(name, knowledge, habitat, topSpeed, color, flying)

Dog

(name, knowledge, habitat, topSpeed, color,bargsAlot)

Whale

(name, knowledge, habitat, topSpeed, color, picture)

En esta sección usaremos las variables instancia **name, vocabulary** y **barksAlot**. La variable instancia **name** contiene un string representando el nombre de el animal, **vocabulary** contiene un string de todos las palabras conocidas por un perico, y **barksAlot** contiene ya sea **true** o **false**, dependiendo de si el perro ladra mucho o no.

Normalmente tu crearías la nueva clase usando el Class Hierarchy Browser. Pero como son muchas las clases y los métodos a definir para la jerarquía de los animales (en este caso), hemos simplificado el

procedimiento para ti poniendo esta definición inicial de la clase en un archivo. Para agregar las clases de los animales a tu ambiente Smalltalk instala el archivo mediante la evaluación de la siguiente expresión:

```
(File pathName: 'tutor\animal6.st') fileIn; close
```

Ahora para ver las nuevas clases con el Class Hierarchy Browser, activa esta ventana, y del menú **Classes** escoge la opción **Update**. Ahora todas las clases de los animales son visibles. Si no es así utiliza la opción **Hide/Show** de el menú **Classes**.

Los Métodos de las clases de los Animales.

Como puedes ver de el Class Hierarchy Browser, los métodos que has incluido para la clase **Animal** son **answer:**, **name:**, y **talk**. El código para estos métodos es el siguiente:

```
        answer: aString
    "Despliega un mensaje para el receptor animal
      sobre la ventana Transcript, consistiendo
      de el nombre de la clase de el animal y un nombre
      precedido de aString."
  Transcript nextPutAll:
      self class name, ' ', name, ': ', aString;
      cr

name: aString
    "Cambia el nombre de el receptor animal a aString."
     name := aString

talk
    "Despliega un mensaje diciendo que el receptor no puede hablar."
     self answer: 'I can"t talk'!
```

```
Animal subclass: #Bird
  instanceVariableNames: 'flying '
  classVariableNames: ''
  poolDictionaries: ''

Bird subclass: #Parrot
  instanceVariableNames: 'vocabulary '
  classVariableNames: ''
  poolDictionaries: ''
```

De manera similar, los métodos para la clase Parrot son:

```
      talk
    "Despliega un mensaje conteniendo el receptor de el  vocabulario de el périco."
      self answer: vocabulary
      vocabulary: aString
    "Cambia el vocabulario de el périco receptor a aString."
      vocabulary := aString
```

Y finalmente, los métodos para la clase Dog: son:

```
  bark
    "Tiene que la ladrar el perro receptor haciendo
      sonar la campana de desplegando un mensaje de
      ladrido."
  Terminal bell.
  barkaAlot
    ifTrue: [self answer: 'Bow Wow, Bow Wow, Bow Wow!!']
    ifFalse: [self answer: 'Woof']!

beNoisy
    "Cambia el status de el perro receptor a ruidoso."
  barkaAlot := true.
  self answer: 'I''ll bark a lot'!
```

```
beQuiet
    "Cambia el status de el perro receptor a silencioso."
  barksAlot := false.
  self answer: 'I won"t bark much'!

talk
    "Tiene que hablar el perro receptor ladrando a menos que
     barksAlos sea nil, en cuyo caso la superclase decide como hablar."
  barksAlot isNil
    ifTrue: [super talk]
    ifFalse: [self bark]
```

No hemos definido ningún método para las clases **Penguin** y **Whale**. Sin embargo, estas clases heredan los métodos de la clase **Animal**, así que podemos crear los objetos **Whale** y **Penguin** y enviarles mensajes a ellos, ya que los utilizaremos más adelante.

Herencia de los Métodos.

Como las variables instancia, los métodos también son heredados. Cuando un mensaje es enviado a un objeto, Smalltalk busca el correspondiente método definido en la clase de el objeto. Si lo encuentra, Smalltalk lo ejecuta. Si no encuentra el método, Smalltalk repite el procedimiento en la superclase de el objeto. Este proceso continúa en todo el camino ascendente hacia la clase **Object**. Si ningún método es encontrado en alguna superclase, una ventana **Walkback** es desplegada en la pantalla, para informar que hay un error.

Por ejemplo, observa el método **name:** definido en la clase **Animal**. Como este método no está definido en ninguna subclase de la clase **Animal**, el método **name:** en la clase **Animal** es evaluada para cualquier mensaje **name:** que sea enviado a las instancias de la clase **Dog**, **Parrot**, **Penguin** o **Whale**.

Como otro ejemplo, observa el método **talk** en la clase animal. Las clases **Penguin** y **Whale** heredan **talk** de la clase **Animal**, mientras las clases **Dog** y **Parrot** reimplementan sus propias versiones de **talk**.

La herencia de métodos de la jerarquía animal es mostrada en la siguiente figura:

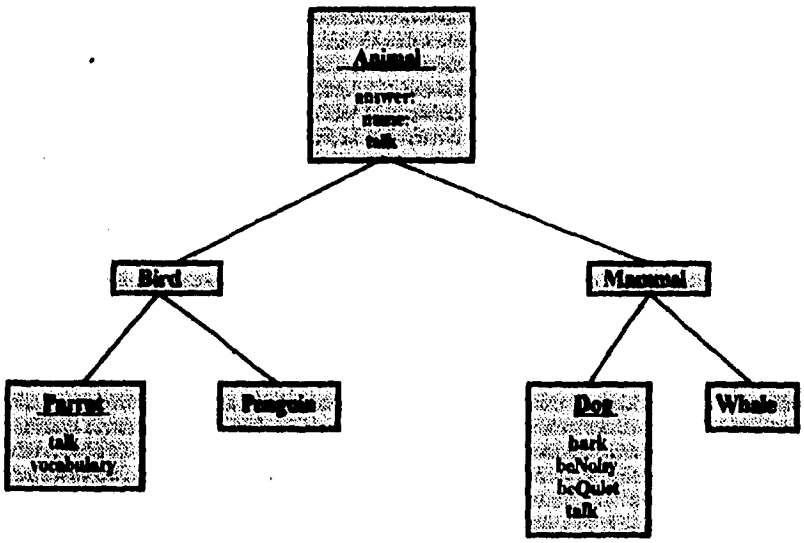


Figura 4. 3 La herencia de Métodos de la Jerarquía de Animales.

La variable especial "Super".

Ocasionalmente, tu querrás no ejecutar un método y utilizar el método de más arriba en la cadena de la jerarquía, es decir, el de su superclase. Por ejemplo, observa el método **talk** de la clase **Dog**. Un perro no sabe como ladrar si su variable instancia **barksAlot** esta indefinida (tiene el valor **nil**). En este caso, este usa el siguiente mensaje para preguntar por el método **talk** de su superclase :

super talk

La variable especial **super** representa el mismo objeto como la variable especial self -el receptor en el método en el cual este aparece. La diferencia es que cuando un mensaje es enviado a **super**, Smalltalk/V comienza a buscar por el método no en la clase de el objeto receptor, sino en la superclase de la clase que contiene el método en el cual **super** aparece. En el ejemplo del método **talk**, la búsqueda comienza en **Mammal**,, la superclase de **Dog**. No hay método **talk** en **Mammal**, pero si hay en la clase **Animal**, así que este es usado.

Creando los objetos de los Animales.

Evalúa la siguientes expresiones (con **Do It**) para crear y asignar cinco variables globales, los objetos animales: dos perros, un pingüino, un périco y una ballena (los animales "hablaran" en la ventana Transcript, así que arregla las ventanas para que no traslapen a la ventana Transcript):

```
Snoopy := Dog new.
Snoopy name: 'Snoopy'.
Snoopy beQuiet.
Lassie := Dog new.
Lassie name: 'Lassie'.
Lassie beNoisy.
Wally := Penguin new.
Wally name: 'Wally'.
Polly := Parrot new.
Polly name: 'Polly'.
Polly vocabulary: 'Polly want a Cracker'.
Moby := Whale new.
Moby name: 'Moby'
```

Polimorfismo

Polimorfismo es la única característica de la programación orientanda a objetos donde diferentes objetos responden a el mismo mensaje con sus propios y únicos comportamientos. Por ejemplo, evalúa las siguientes expresiones para ver como los diversos animales responden a el mensaje **talk**:

```
Lassie talk.
Snoopy talk.
Wally talk.
Polly talk; talk; talk.
Polly vocabulary: 'Screeech(@#!? Don''t bother me!'.
Polly talk.
Moby talk.
Snoopy beNoisy; talk.
Lassie beQuiet; talk
```

CAPITULO 5 Flujos, Colecciones y Depuración

Flujos (Streams)

Smalltalk/V soporta muchos diferentes tipos de objetos stream. Ya has visto alguno de ellos cuando accesaste los archivos de disco usando los objetos **FileStream**. Los stream son también usados para accesar colecciones de objetos internos, tales como strings y arreglos usando las clases **ReadStream**, **WriteStream**, y **ReadWriteStream**. Las clases stream son arregladas en una jerarquía con la clase **Stream** como la superclase de todas las demás. Puedes usar el Class Hierarchy Browser (explicado anteriormente) para explorar esta jerarquía.

Este capítulo presentará una serie de ejemplos usando streams, los cuales te darán una buena introducción.

Los streams son frecuentemente usados para buscar entradas o producir salidas editadas. Por ejemplo, observa este método, el cual realiza ambas operaciones:

```
| input output char fecha |
fecha := Date today printString.
input := ReadStream on: 'La fecha de hoy es %'.
output := WriteStream on: String new.
[input atEnd]
  whileFalse: [
    (char := input next) = $%
      ifTrue: [output nextPutAll: fecha]
      ifFalse: [ output nextPut: char ] ].
^output contents
```

Este ejemplo crea dos streams. El mensaje on: es enviado a la clase **ReadStream** para crear un stream sobre el argumento string **'La fecha de hoy es %'**. The mensaje on: es también usado para crear un **WriteStream** sobre un string vacío, para almacenar la salida editada.

Como los nombres lo implican, **ReadStream** puede sólo ser leído y **WriteStream** puede sólo ser escrito. Como hemos visto previamente en los ejemplos de los ejemplos de disco, los streams son leídos con el mensaje next y escritos con el mensaje nextPut:. El mensaje atEnd prueba si hay más entrada

para ser leída. El mensaje **nextPutAll:** escribe diversos objetos a un stream a la vez.. En el ejemplo anterior, el argumento es un string de caracteres conteniendo la fecha de hoy.

En el ejemplo de arriba el stream es aplicado sobre strings de caracteres. Este usa un string vacío para el **WriteStream** porque los streams automáticamente crecen como sea necesario, para acomodar los objetos escritos en él. El contenido de el mensaje retorna un string conteniendo todos los objetos escritos a el stream.

Para cambiar el ejemplo de arriba, con el propósito de usar archivos de disco en vez de streams sobre cadenas, simplemente se cambia el mensaje que crea los streams **input** y **output**. Esto ilustra una de las características más poderosas de Smalltalk/V, puedes escribir programas que son dependientes sobre el comportamiento, sin importar la estructura de datos. Este significa que puedes escribir y probar un programa usando simples objetos internos, tales como streams sobre strings, y entonces fácilmente extenderlo para uso de archivos externos.

Los streams no están restringidos para leer y escribir únicamente sobre caracteres. Por ejemplo, este método lee y escribe un Array de Números objetos:

```
| input output |
input := ReadStream on: #( 1 5 10 20 ).
output := WriteStream on: Array new.
[input atEnd]
   whileFalse: [
      output nextPut: input next factorial].
^output contents
```

Aunque estos ejemplos no lo muestran, los streams pueden también ser reposicionados, tal y como un archivo de acceso aleatorio, usando el mensaje **position:**. El argumento es un entero. Puedes también usar el mensaje **position:** para obtener la posición actual de un stream..

Colecciones

Las colecciones son objetos que contienen una colección de otros objetos. Hasta este momento ya has visto dos tipos de colecciones: **Array** y **String**. Los strings son secuencias de caracteres de tamaño fijo, mientras los arreglos son secuencias arbitrarias de objetos de tamaño fijo. También has usado los mensajes de los iteradores **do:**, **collect:**, **select:**, y **reject:**, con los arreglos y strings. Estos mensajes son entendidos por todas las clases de colección, tres de las cuales son **Dictionary**, **Bag**, y **Set**.

Diccionarios

Los diccionarios almacenan y retraen objetos mediante el uso de una llave. Por ejemplo, creemos un simple directorio de teléfonos. Primero, creemos una variable global conteniendo un Diccionario vacío mediante la evaluación de lo siguiente (contesta Yes a el prompt):

```
LibroTelefonico := Dictionary new
```

Para agregar números a el libro telefónico, utiliza el mensaje **at:put:** :

```
LibroTelefonico
at: 'Marisa' put: '645-1082';
at: 'Francesca' put: '555-1212';
at: 'Jackie' put: '392-481-5000';
at: 'Rakesh' put: '645-1083';
at: 'Vijay' put: '645-1083'
```

En las expresiones de arriba, los strings **'Marisa'** y **'Francesca'** son las llaves y los strings **'645-1082'** y **'555-1212'** son los valores correspondientes. Nota que **at:put:** es también usado para accesar los elementos de Strings y Arreglos. Con los objetos del Diccionario, sin embargo, el primer argumento es la llave en el Diccionario, en vez de la posición en el Arreglo o en el String.

Para retraer un objeto de un Diccionario, usa el mensaje **at:** con la llave como el argumento. Por ejemplo, la siguiente expresión retorna el string **'645-1082'**:

```
LibroTelefonico at: 'Marisa'  '645-1082'
```

Para probar si un objeto existe como una llave en un Diccionario, usa el mensaje **includeKey** en la siguiente expresión:

```
(LibroTelefonico includesKey: 'Aaron')
ifTrue: [LibroTelefonico at: 'Aaron']
ifFalse: ['No esta en el Libro Telefónico']
```

Un simple camino para hacer esto es usar el mensaje **at:ifAbsent:**. El primer argumento es la llave y el segundo argumento es un bloque de código que será ejecutado si las llaves no están en el Diccionario receptor. Por ejemplo:

```
LibroTelefonico at: 'Aaron' ifAbsent: ['No esta en el Libro      Telefónico']
```

Las llaves y los valores almacenados en un Diccionario pueden ser cualquier tipo de objetos.

Los Diccionarios son tan útiles que tienen una ventana de inspector especial llamada el Inspector de Diccionarios. Para abrir un Inspector Dictionary sobre el libro de teléfonos, avalúa la siguiente expresión:

```
LibroTelefonico inspect
```

Puedes también hacer doble click sobre la palabra **LibroTelefonico** y seleccionar **Inspect It** de el menú Smalltalk. El recuadro de la derecha de la ventana es una lista ordenada de todos las llaves en el Diccionario, en nuestro caso los nombres de las personas en el libro telefónico. Cuando seleccionas una llave, el valor correspondiente es desplegado en el recuadro de la derecha, en nuestro caso el número de teléfono de la persona. Mediante el uso de los menús de Edición e Inspección, las entradas pueden ser editadas, agregadas y removidas.

Bolsas (Bags)

Las bolsas almacenan un arbitrario número de objetos de cualquier tipo. Al contrario de los Arrays, no existe un orden implícito o secuencia a los elementos (objetos) dentro de la bolsa. Los elementos son agregados a una Bolsa con el mensaje **add:**. Para probar si un objeto esta en una Bolsa, usa el mensaje **includes:**. Por ejemplo, esta expresión lee un archivo y reporta la frecuencia con la cual cada letra ocurre:

```
| input respuesta f c |
input := File pathName: 'tutor\capitulo.5'.
respuesta := WriteStream on: String new.
f := Bag new.
[input atEnd]
        whileFalse: [
                (c := input next) isLetter
                        ifTrue: [f add: c asLowerCase]].
0 to: 25 do: [:i |
        c:= ($a asciiValue + i) asCharacter.
        respuesta
                cr; nextPut: c; space;
                nextPutAll: (f occurrencesOf: c) printString].
^respuesta contents
```

Conjuntos (Sets)

Un conjunto, como una Bolsa, almacena objetos arbitrarios. La diferencia es que un **Set** no almacena el mismo objeto más de una vez. Por ejemplo, esta expresión computa el conjunto de caracteres que ocurren en un archivo y no en el otro:

```
| conjunto1 conjunto2 |
conjunto1 := Set new.
conjunto2 := Set new.
(File pathName:'turor|capitulo.5') do: [:c | conjunto1 add: c].
(File pathName:'turor|capitulo.4') do: [:c | conjunto2 add: c].
^conjunto1 reject: [:c | conjunto2 includes: c]
```

El mensaje **asSet** crea un conjunto de la colección de objetos de el receptor. Esta una buena forma de eliminar duplicados de una **Collection**. Por ejemplo, para computar las únicas vocales en un string, evalúa lo siguiente:

```
'Ahora es el tiempo' asSet select: [:c | c isVowel]
```

Depuración (Debugger)

Esta sección utiliza lo que has aprendido en secciones pasadas para construir una completa aplicación usando colecciones y streams. El programa, sin embargo, contiene diversos errores a propósito. Así es como adquirirás alguna experiencia localizando y corrigiendo errores usando el depurador (Debugger) de Smalltalk/V.

Un Sistema de Carga de Documentos.

La primera cosa que hay que hacer es implementar una nueva clase, **WordIndex**, la cual te permite crear una base de datos de documentos y localizar estos basándose en las palabras que contienen. Los Documentos son archivos de texto en ASCII, vistos como una serie de palabras conteniendo caracteres alfanuméricos separados por una serie de caracteres no alfanuméricos.

Cuestionaras a la base de datos proveyéndola de una colección de strings de palabras. La pregunta retornará una colección de nombres de archivo de todos los documentos que contienen todas las palabras.

Las instancias de la clase **WordIndex** tienen las variables instancia **documents** y **words**:

- **documents** es un conjunto de strings de la ruta de el archivo del documento cuyas palabras hayan chocado con el índice de palabras.
- **words** es un diccionario, con cada llave conteniendo un string para una palabra y cada valor será un conjunto conteniendo los rutas de todos los documentos que contiene la palabra.

Por lo tanto, la definición de la clase es:

```
Object subclass: #WordIndex
instanceVariableNames:
'documentos palabras '
classVariableNames: ''
poolDictionaries: ''
```

Agrega la definición de la clase **WordIndex** y sus métodos a la imagen de Smalltalk/V por la evaluación de la siguiente expresión:

```
(File pathName: 'tutor\wrdindx8.st') fileIn; close
```

Ahora selecciona la opción **Update** del menú **Classes** de el Class Hierarchy Browser de manera que puedas observar los métodos de la nueva clase **WordIndex**. Hay seis métodos definidos por la clase **WordIndex**, que son los siguientes:

```
addDocument: pathName
"Agrega todas las palabras en el documento descrito por
la ruta string a las palabras de el diccionario."
    | palabra palabraStream |
    (documentos includes: pathName)
    ifTrue: [self removeDocument: pathName].
    palabraStream := File pathName: pathName.
    documentos add: pathName.
    [(palabra := palabraStream nextWord) == nil]
    whileFalse: [
        self addWord: palabra asLowerCase to: pathName].
        palabraStream close
```

```
addWord: palabraString for: pathName
"Agrega la palabra a el diccionario de palabras para
el documento descrito por pathName."
(palabras at: palabraString) add: pathName

initialize
"Inicializa un nuevo WordIndex vacío."
documentos := Set new.
palabras := Dictionary new

locateDocuments: queryWords
"Responde un arreglo de las rutas para
todos los documentos los cuales contienen
todas las palabras en queryWords."
| respuesta bolsa |
bolsa := Bag new.
respuesta := Set new.
queryWords do: [ :palabra |
bolsa addAll:
(documentos at: palabra ifAbsent: [ #( ) ] ) ].
bolsa asSet do: [ :documento |
queryWords size =
(bolsa occurrencesOf: documento)
ifTrue: [ respuesta add: documento ] ].
^respuesta asSortedCollection asArray

removeDocument: pathName
"Elimina el string de ruta que describe un
documento de el diccionario de palabras."
palabras do: [ :docs | docs remove: pathName ].
self removeUnusedWords!
```

```
removeUnusedWords
    "Elimina todas las palabras la cuales tengan
     una collection vacía de documento."
        | nuevasPalabras |
        nuevasPalabras := Dictionary new.
        palabras associationsDo: [ :unaAsoc |
        unaAsoc value isEmpty
      ifFalse: [ nuevasPalabras add: unaAsoc ] ].
      palabras := nuevasPalabras
```

Como trabaja la clase WordIndex.

Veamos como trabaja la clase **WordIndex** en términos de mensajes de alto nivel los cuales crean un índice y hacen preguntas.

Mencionamos antes que se habían colocado algunos errores intencionales, este es el primer lugar donde ocurrirán. Por esta razón, ******* no evalúes los mensajes hasta que el tutor te lo indique.******

Construye y usa el índice de palabras en tres pasos. Primero, crea un índice de palabras vacío (recuerda, no evalúes esta expresión todavía):

```
Indice := WordIndex new initialize
```

Indice es creado como una nueva variable global la cual inmediatamente será puesta dentro de un índice de palabras. El método **addDocument:** crea un stream de archivo para buscar el documento, repetidamente envía el mensaje **nextWord** para introducir cada par de palabras documento en el diccionario **palabras**. Por ejemplo, para agregar las palabras de los archivos de ejemplos de los capítulos 4 y 5, tú utilizarás las siguientes expresiones (nuevamente, no evalúes estas todavía):

```
Indice addDocument: 'tutor\capitulo.5'.
Indice addDocument: 'tutor\capitulo.4'.
```

Para interrogar a el índice de palabras, usa el mensaje **locateDocuments:**, como en los siguientes ejemplos (nuevamente, no los evalúes aún).

```
Indice locateDocuments: #('show' 'class')
Indice locateDocuments: #( 'where' 'the' 'turtle')
Indice locateDocuments: #( 'each' 'talk')
```

Cada pregunta de arriba retoma un arreglo de strings, conteniendo la ruta del documento para dos de los documentos que contienen todas las palabras en la interrogación.

El método **locateDocuments:** es algo más complejo que los otros métodos en su clase. Este usa una bolsa para acumular todos las rutas de todos los archivos que contienen cada palabra en la interrogación. (Recuerda que las bolsas (bags), al contrario de los conjuntos (sets), pueden contener múltiples ocurrencias de el mismo objeto). Este entonces examina la bolsa para encontrar cualesquiera documentos los cuales son repetidos tantas veces como haya palabras en la interrogación; existen documentos que contienen todas las palabras.

Depurando la clase WordIndex .

Así es como se supone debe trabajar la clase. Veamos si lo hace. (De este punto en adelante, comienza a evaluar las expresiones de ejemplo nuevamente). Primero, construye una nueva palabra de índice y asignala a la variable global indice

```
Indice := WordIndex new initialize
```

Ahora agrega el archivo tutor capitulo 4 y 5 mediante la evaluación de los siguientes mensajes **addDocuments:** .

```
Indice addDocument: 'tutor\capitulo.5'.
Indice addDocument: 'tutor\capitulo.4'.
```

Que pasa! En lugar de agregar los archivos del tutor, hemos obtenido una ventana **Walkback**.

Como viste en el capitulo 2, una ventana **Walkback** describe una condición de error. El titulo de la ventana muestra la condición de el error, y el texto muestra los mensajes enviados más recientemente, con aquellos más recientes apareciendo primero.

En la actual ventana **Walkback**, la etiqueta dice que el mensaje **addWord:to:** no es entendido, mientras en la linea más alta de el texto se muestra **WordIndex** como la clase de el objeto el cual no entiende el mensaje.

No importa donde obtengas una ventana **Walkback**, generalmente harás una de tres cosas:

- Puedes determinar que problema es a partir de la información contenida en la ventana **Walkback**. En este caso, normalmente cerrarías la ventana y entonces corregirías el problema, usualmente a través de el Class Hierarchy Browser.
- Puedes determinar que la ventana Walkback ocurrió como un resultado de que teclearas **Ctrl+Break** simultáneamente, o porque una mensaje **halt** fue enviado. En este punto, puedes seleccionar ya sea **Resume** o **Debug** de el menú de la ventana **Walkback**. En este caso, no hay problema con el programa, y si tu seleccionas **Resume**, la ventana **Walkback** se cierra y la ejecución continua.
- Puedes decidir que necesitas más información y te gustaría usar el Depurador para obtenerla. Selecciona **Debug** en el menú **Walkback**. La ventana **Walkback** se cierra y la ventana de Depuración aparece.

Para el caso que estamos tratando, tu tienes suficiente información en la ventana **Walkback** para resolver el problema. Observa el código para la clase **WordIndex** usando el Class Hierarchy Browser. Nosotros definimos un método **addWord:for:**, pero enviamos el mensaje **addWord:to:** (en el método **addDocumet:**) el cual no fue entendido. Usamos un mal mensaje! Cierra la ventana **Walkback**.

Corrige el método **addDocument:** para usar **addWord:for:** en vez de **addWord:to:**. Entonces trata de agregar los archivos del tutor a el índice de palabras usando las siguientes expresiones:

```
Indice addDocument: 'tutor\capitulo.5'.
Indice addDocument: 'tutor\capitulo.4'.
```

No esta corregido aún!. Esta vez, obtuviste una nueva ventana **Walkback**. El título de la ventana dice **Key is missing**. Como el problema no es obvio, puedes obtener más información mediante la apertura de la ventana de Depuración. Selecciona **Debug** de el menú o presiona el botón **Debug** en la sección del texto. La ventana **Walkback** se cierra y verás la figura 5.1. Observala.

Mecánica de uso del Depurador

La ventana del Depurador, con sus múltiples recuadros, da una vista expandida de el estado de el ambiente Smalltalk el cual fue sumarizado en el **Walkback**. El recuadro superior izquierdo (una lista) repite la información de el **Walkback**; puedes utilizar en este recuadro alguna línea. Cuando seleccionas una línea, los otros recuadros son vaciados con la información relacionada.

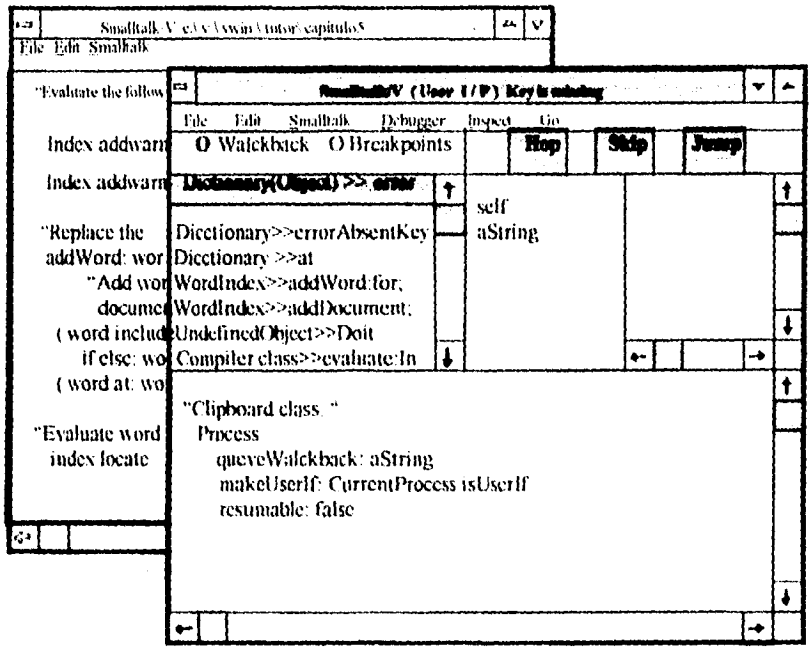


Figura 5.1 El depurador y sus botones.

Selecciona la línea conteniendo **Dictionary>>at:** En el recuadro de texto de abajo se muestra el código fuente para el método seleccionado, en este caso **at:** de la clase Dictionary

```
at:aKey
"Responde el valor de el par de llaves o par de
  valores cuyas llaves sean igual a la llave de
  el receptor. Si no se encontró, reporta un error."
| respuesta |
^(respuesta := self lookUpKey: aKey) == nil.
        ifTrue: [self errorAbsentKey]
        ifFalse: [respuesta value]
```

El texto que esta marcado es la expresión que en este momento esta siendo evaluada en este método.

El método **at:** responde el valor de el valor de el par de llaves cuya llave sea igual a **aKey** en el diccionario receptor. Si la llave esta pérdida (Key is missing), **errorAbsetKey** es invocado el cual originó la ventana **Walkback**.

Los dos recuadros de arriba a la derecha son un inspector para el receptor, argumentos y variables temporales de el método seleccionado. En este caso, verás el receptor **self**, el argumento **aKey** y la variable temporal **answer**. Selecciona **self**; verás que el valor es un diccionario vacío. Ahora selecciona **aKey**. El valor es el string **'ejemplos'**, la primera palabra en el archivo. Tratamos de hacer buscar a el Diccionario en un diccionario vacio, **self**, con la primera palabra en al archivo como llave

Selecciona la linea contenendo **AddWord:for:** en el recuadro de **Walkback** sobre la parte superior izquierda de la ventana. Ahora selecciona el argumento **wordString**. Nuevamente, esta el string **'ejemplos'**. Tratamos de accesar el Diccionario **words** con una llave, sin primero probar si alguna llave esta presente! Corrige el método **addWord:for:** en el recuadro de abajo de el Depurador como se muestra:

```
addWord: wordString for: pathName
"Agrega wordString a el diccionario de palabras
para el documento descrito por la ruta pathName."
(words includesKey: wordString)
        ifFalse: [words at: wordString put: Set new].
(words at: wordString) add: pathName
```

Selecciona **Save** de el menú File o presiona **Alt+S**. Nota que pasa. Las lineas arriba de **addWord:for:** en la lista **Walkback** son descartadas porque un método que ellos retornaron ya ha sido cambiado. El método **addWord:for:** permanece seleccionado. Ahora selecciona **Restart** de el menú del Depurador La ejecución se reinicia mediante el reenvio de el mensaje seleccionado.

La ventana del Depurador desaparece y el método construye el indice. Con el diccionario ahora ya construido, prueba hacer algunas preguntas. Evaluando la expresión:

```
Indice locateDocuments: #('show' 'class')
```

Otro **Walkback** aparece señalando otro error. Inmediatamente abre la ventana del Depurador sobre este error.

El mensaje **at:ifAbset:** fue enviado a una instancia de la clase **Set** la cual no lo entendió. Selecciona la linea del **Walkback** que contiene **Set(Object)>>doesNotUnderstand:** . Entonces selecciona **self** en la lista de variables temporales. El valor es:

Set('tutor\capitulo.5' 'tutor\capitulo.4')

Ahora selecciona la tercera linea de el Walkback, representando un bloque en el método **locateDocumets:**, y examina los valores de las variables temporales Observa el código fuente de el método. El mensaje **at:ifAbsent:** que esta siendo ejecutado aparece marcado. Este usa la variable instancia **documents** como receptor. El valor impreso para el conjunto de arriba confirma esto, porque este no contiene las rutas de los documentos.

Considera esta sentencia. Ya sea que enviemos el mensaje equivocado a **documents** o **documents** es el receptor equivocado. Esta sentencia esta tratando de agregar a la variable **bag** todos los documentos que incluyen el string contenido en la variable **word**. El receptor esta indicado como erróneo. Esta sentencia se usará en vez de el diccionario **words**:

```
bag addAll:
        (words at: word ifAbsent: [#()])
```

Cambia el método **locateDocuments:** usando el recuadro de texto en el Depurador, salva el cambio del método, y reinicia en **locateDocuments:**. Ahora si trabaja!
Prueba las siguientes preguntas:

```
Indice locateDocuments: #( 'where' 'the' 'turtle')
Indice locateDocuments: #( 'each' 'talk')
```

Hop, Skip y Jump.

Ahora que ya tienes la clase **WordIndex** depurada, veamos como puedes usar el Depurador para aprender como opera una aplicación mediante la observación de el envío de mensajes. Abre una ventana del Depurador para observar la ejecución de la interrogación que has ejecutado paso por paso mediante la evaluación de la siguiente expresión:

```
        self halt.
    Indice locateDocuments: #( 'each' 'talk')
```

Existen 5 botones de izquierda a derecha justo abajo de la barra de titulo de la ventana; los radiobotones **Walkback** y **Breakpoints** y los push botons **Hop**, **Skip** y **Jump**. **Hop**, **Skip** y **Jump** cada uno causa una limitada ejecución de el programa.

- **Hop** ejecuta el menor incremento: un mensaje Smalltalk es enviado o una sentencia de asignación.
- **Skip** ejecuta más que **Hop**: va a el siguiente mensaje enviado o una asignación en el método actual o llega al siguiente breakpoint, lo que suceda primero.
- **Jump** ejecuta más que **Skip**: arriba a el siguiente breakpoint o a el final de la expresión a depurar.

Prueba el botón **Hop** dos veces y ve lo que aparece en la ventana del Depurador. El estado de la ejecución se encuentra en el comienzo de la ejecución de la expresión. Presiona **Hop** nuevamente. Nota que la ejecución se hace en pequeños incrementos con la siguiente sentencia a ser ejecutada marcada después de el avance. Puedes examinar el estado de los objetos después de cada **Hop**.

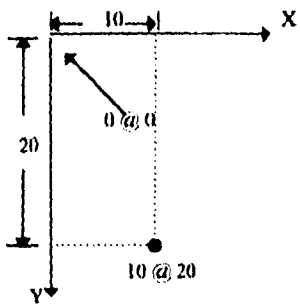
Ahora prueba presionando el botón **Skip** varias veces. Nota los diferentes estados dentro de el mismo método hasta que el método finaliza su ejecución. Esto te permite concentrarte sobre la activación de un sólo método e ignora los mensajes de menor nivel.

Jump toma los más grandes pasos de los botones de ejecución por pasos. En este caso, **Jump** progresará a el final de la expresión depurada ya que no se ha establecido ningún breakpoint.

CAPITULO 6 Manejo de Gráficos y Ventanas

Coordenadas en la Pantalla.

Para referirnos a una coordenada especifica en la pantalla, nos apoyamos en el concepto de "punto", que en Smalltalk/V queda definido como un par que consiste de una coordenada X y otra coordenada Y. Como ejemplo podemos escribir 10 @ 20 que denota un punto en la pantalla con una coordenada X igual 10 y otra coordenada Y igual a 20. La siguiente figura muestra el sistema coordenado que utiliza Smalltalk para posicionar y mostrar puntos en pantalla:



Nota que hay un punto origen denotado por la coordenada 0 @ 0 en la esquina superior izquierda. A partir de este punto la coordenada en X aumenta su tamaño hacia la derecha, y la coordenada en Y aumenta su tamaño hacia abajo. El eje X es una línea imaginaria que empieza a crecer a partir de la coordenada Y cero. En forma similar, la linea imaginaria Y crece a partir de la coordenada X cero.

Evalúa las siguientes expresiones:

```
5 @ 10
```

y ahora estos ejemplos más:

```
(5 @ 10) x
(5 @ 10) y
```

Estas expresiones retoman los valores de 5 y 10, respectivamente. También se puede transportar, multiplicar, dividir, o comparar puntos, como en los siguientes ejemplos:

```
(1 @ 2) rightAndDown: (1 @ 2)
```

En este ejemplo el objeto rightAndDown nos desplazará al punto (2 @ 4), ya que a la coordenada en X aumenta un valor de 1 y a la coordenada en Y aumenta un 2.

(1 @ 2) leftAndUp: (1 @ 2)

Esta es la situación contraria a la anterior, ya que ahora se resta un 1 a la coordenada X y un 2 a la coordenada Y. Esto implica quedar posicionados en la coordenada (0 @ 0).

(1 @ 2) * (1/2 @ (1/2))

Este es un ejemplo de multiplicación, donde 1 por (1/2) nos da un (1/2) y 2 por (1/2) nos da 1; esto implica quedar posicionados en el punto (1/2 @ 1).

(1 @ 2) < (3 @ 4)

Este es un ejemplo de comparación donde 1 es menor que 3 y 2 es menor que 4. Esto implica que el resultado de esta evaluación de por resultado verdadero (true).

(3 @ 5) > (3 @ 4)

Este es otro ejemplo de comparación donde 3 no es mayor que 3, aunque 5 si sea mayor que 4. El resultado de esta evaluación da por resultado falso (false), ya que no se cumplen las dos comparaciones.

También es posible mezclar puntos con escalares:

(1 @ 2) + 1

En este ejemplo el escalar 1 se suma a ambas coordenadas X y Y. En consecuencia tendremos el punto final 2 @ 3.

(1 @ 2) // 2

En este ejemplo el escalar 2 funciona como divisor para ambas coordenadas del punto 1@2. Esto implica que tendremos como resultado el punto 1/2 @ 1.

Hay ciertas expresiones de puntos y escalares que no son posibles realizarlas. Como ejemplo evalúa la siguiente expresión:

(1 @ 2) < 3

Esta es una operación no válida, ya que un punto esta formado de dos coordenadas que no puede ser afectado por el escalar.

Para alterar una de las dos coordenadas, simplemente usamos los mensajes X y Y. Por ejemplo, evalúa la siguiente expresión como un grupo:

```
| punto |
punto := (5 @ 10).
punto x: 1.
punto y: 2.
^punto
```

Un Rectángulo.

En general, para dibujar algo tan simple como un rectángulo, requerimos utilizar una pluma (en relación con el objeto Display), una posición apropiada y entonces mover esto cuatro veces para dibujar los lados.

Además para ser capaces de dibujar, el objeto **pen** debe estar en estado **down**. Para asegurar esto debemos ejecutar la siguiente expresión:

```
Display pen down
```

Nota que estamos obteniendo la pluma (pen) del Display y entonces le enviamos el mensaje **down**. En general podemos adoptar la convención de que siempre que usemos una pluma, su estado por default sea **down** para que cuando usemos una pluma no sea necesario estar ejecutando la expresión anterior cada vez.

Para nuestro primer ejemplo, consideremos dibujar un cuadrado con un ancho de 100 pixeles. Además, podemos usar una variable para referirnos a nuestra pluma (**pen**). Consecuentemente, será fácil usar la pluma para dibujar cuatro lados en la pantalla:

```
| pluma |
pluma := Display pen.
pluma place: 100 @ 100.    " Se inicia el punto "
pluma goto: 200 @ 100.     " Fijamos a la derecha un ancho de 100 pixeles "
pluma goto: 200 @ 200.     " Fijamos 100 pixeles hacia abajo "
pluma goto: 100 @ 200.     " Fijamos 100 pixeles a la izquierda "
pluma goto: 100 @ 100.     " Fijamos 100 pixeles hacia arriba "
```

Si necesitamos dibujar un cuadrado de diferente tamaño o el mismo cuadrado en coordenadas diferentes, debemos hacer cambios substanciales al bloque de instrucciones anterior. Esto resulta simple si usamos variables que contengan los cambios necesarios. Iniciaremos agregando dos nuevas variables (para propósitos de comparación).

Nota: Es muy importante que te des cuenta que en el siguiente ejemplo se están utilizando las definiciones anteriores, sobre operaciones con puntos y coordenadas. Esta habilidad es la que nos permite realizar programas de una forma más depurada y eficiente.

```
| pluma inicio area |
pluma := Display pen.
inicio := 300 @ 300.        " Se inicia el punto "
area := 100.                " 100 pixeles "
pluma
place: inicio;              " Se define la esquina superior izquierda del cuadrado "
goto: inicio + (area @ 0);  " Se define la esquina superior derecha del cuadrado "
goto: inicio + area;        " Se define la esquina inferior derecha del cuadrado "
goto: inicio + (0 @ area);  " Se define la esquina inferior izquierda del cuadrado "
goto: inicio.               " Se define la esquina superior izquierda nuevamente "
```

Mapeo de bits

Un objeto que puede mantener una imagen gráfica es una instancia de la clase **Bitmap**.Un bitmap es un arreglo de bits que representan una imagen. Cuando se crea una imagen monocromática, los bits 0 y 1 son lo único que se necesita. Por el contrario si deseamos manipular una imagen que contiene color, se deben usar 4, 8 o 24 bits por pixel que ocasiona un aumento en la complejidad de la estructura del bitmap. El mapeo de bits es muy importante, porque con su uso es posible desplegar imágenes que de alguna otra manera sería algo complicado.

En el próximo ejemplo se crea un **bitmap** y se dibuja dentro de él con una pluma (**pen**). El bitmap en memoria actúa como un dispositivo físico, tales como la pantalla o la impresora. El ejemplo, usa entonces la expresión **Display pen** y el método **DisplayAt**: para dibujar el bitmap sobre la pantalla en el punto 10 @ 100:

```
| bitmap |
bitmap := Bitmap screenExtent: 100 @ 100.
"creamos un Bitmap con los mismos colores por pixel como la pantalla"
bitmap pen
  home;
  fill: ClrRed;
  foreColor: ClrGreen;
  ellipse: 40 minor: 20;
  displayText: 'Hola'.
  bitmap displayAt: 10 @ 100 with: Display pen.
  bitmap release  "Desactivamos el bitmap de la memoria"
```

Observa la línea de código final, donde se libera el bitmap de la memoria. Esto debe hacerse siempre.

Herramientas gráficas

Todos los objetos que se usan en Smalltalk/V para dibujar se agrupan en la clase llamada **GraphicsTool**. Esta clase contiene variables y métodos comunes a todas las herramientas gráficas. GraphicsTool tiene varias subclases que extienden la gran funcionalidad de esta superclase. Por ejemplo, una instancia de **TextTool** trabaja como máquina de escribir y únicamente despliega caracteres. Una instancia de la subclase **pen** (vista anteriormente) es capaz de escribir, y también de dibujar gráficos. Una instancia de de la subclase **RecordingPen** tiene las capacidades del **Textool** y **Pen**, además de poder recortar gráficos para colocarlos posteriormente en otro sitio.

Textool

La clase **TextTool** trabaja como una máquina de escribir excepto, que como hereda métodos de GraphicsTools, estos le permiten rellenar regiones y mover bloques. Una instancia de **Textool** aparece asociada normalmente con una instancia de **TextPane**.

La dirección de una variable heredada por un **TextTool** es el punto base como primer carácter en una cadena que va a dibujarse. En Smalltalk/V, la posición es guardada por el sistema operativo, pero se pueden enviar mensajes para mover y preguntar por la posición del **TextTool**.

En el siguiente ejemplo utilizamos una ventana de texto (**TextWindow**) y usaremos la clase **TextTool** para desplegar caracteres en diferentes posiciones en una ventana:

```
| aWindow textTool |
aWindow := TextWindow windowLabeled: 'Ejemplo de textTool'
   frame: ( 100 @ 100 extent: 400 @ 200 ).
aWindow nextPutAll: 'Hola!' ; cr.  "desplegamos texto inicial"
textTool := aWindow pane pen. "accesamos la clase textTool"
textTool place: 24 @ 16;          "fijamos la posición"
   displayText: 'Bienvenido'.     "desplegamos una cadena en la posición"
textTool displayText: 'a'
   at: textTool location + (50 @ 30).  "abajo y a la derecha de la posición"
textTool centerText: 'Smalltalk/V'
   at: textTool extent // 2.          "desplegamos una cadena al centro"
textTool lineDisplay: 'Windows'
   at: ( textTool width - 60) @ (textTool height - 20 ).
 " en la esquina derecha inferior y rellenamos con espacios en blanco el resto de la
  línea"
```

Plumas (Pens).

Los objetos pen (plumas) en Smalltalk/V pueden dibujar de un punto a otro sobre la pantalla. En forma general una pluma es una herramienta gráfica que permite dibujar sobre un medio, tal como la pantalla o la impresora.

Los **pens** en Smalltalk/V tienen las caracteristicas siguientes:

- Un plumin, una punta que dibuja cuando su estado es **down** (bajo) y que no lo hace cuando su estado es **up** (elevado).
- Una dirección especificada como un punto
- Una dirección especificada como un ángulo relativo con respecto al eje X, positivo en el sentido de las manecillas del reloj y negativo en sentido antiorario. Así una instrucción de 0° apunta a la derecha, 90° apunta hacia abajo, y -90° hacia arriba.

Adicionalmente un objeto **pen** puede colocarse en un lugar arbitrario o dirigirse a un punto especifico. Para el último caso, una línea se dibuja de la localidad actual al punto destino si el estado del plumin es down. El objeto **pen** puede también ir hacia una distancia arbitraria en alguna dirección partiendo de su posición actual. Cada unidad de distancia, sobre el eje X o el eje Y, corresponde a un **pixel** (punto descrito sobre la pantalla).

Display.

El objeto **Display** se refiere a una área sobre la pantalla que tiene un ancho **(width)** y una altura **(height)** basados sobre un punto que se le llama extendido **(extent)**; la coordenada X del **extent** es el ancho y la coordenada Y es el alto. Esta información puede ser importante cuando se dibuja porque no todos los objetos **Display** tienen el mismo tamaño. Si se desea dibujar una pintura en la esquina inferior derecha, por ejemplo, será necesario conocer el tamaño del extendimiento del **Display**.

Para llevar un orden razonable, mostramos a continuación una lista de todas las operaciones disponibles para los **puntos, plumas (pens)** y para el **Display**.

operaciones sobre números

unNúmero @ unNúmero .....Obtenemos un nuevo punto

operaciones sobre puntos

unPunto x .....Obtenemos la coordenada x del punto

unPunto y .....Obtenemos la coordenada y del punto

operaciones sobre Pens (plumas)

aPen down .....**Habilita** el plumin para dibujar

aPen up .....Deshabilita el plumin para dibujar

aPen north .....Apunta hacia el limite del área

aPen direction ..... Inicia ángulos en grados donde 0° apunta a la derecha, 90° hacia abajo.

aPen **direction**:grados ..... Fija el ángulo donde 0° apunta a la derecha, 90° hacia abajo.

aPen turn:grados .....Reconoce los grados especificados; en sentido horario.

aPen go:distancia .....Mueve un número de pixeles, si la pluma esta en down

aPen home .....Posiciona la pluma al centro sobre el display

aPen location .....Inicia la posición como un punto

aPen goto: aPoint .....Se mueve al punto especificado; dibuja si la pluma esta en estado down

aPen place:aPoint .....Se mueve al punto especificado sin dibujar

operaciones sobre Display

Display pen .....Se genera una pluma para dibujar

Display extent .....Se genera la posibilidad de extender el área del display en base al ancho y altura

Ventanas

A continuación vamos a describir algunas herramientas que nos van a permitir desarrollar software con una presentación rápida y eficiente.

Ya que Smalltak/V es un sistema interactivo, provee facilidades para preguntar, y desplegar información sobre la pantalla. Permitanos empezar a trabajar con estas características.

El Prompter

Las facilidades de los prompters permiten a un usuario suministrar información textual. A continuación desarrollamos un ejemplo que nos ayude a entender su funcionamiento. Para comenzar supongamos que nosotros necesitamos preguntar por el nombre de una persona. Para esto utilizaremos el **prompter** de Smalltalk/V, resultando la siguiente expresión:

```
Prompter prompt: '¿ Cúal es tu nombre?'
    default: 'Hugo.'
```

Al ejecutar esta expresión aparecerá una ventana que te pregunta ¿cúal es tu nombre?. Nota que la expresión pasada tiene como **default** el nombre de Hugo, que aparecerá automáticamente como respuesta. Además la ventana que aparece consta de dos botones. Uno de ellos te señala el **OK** y el otro te muestra la palabra "**Cancel**". Haz click con el mouse en ambos para que observes que sucede.

Una variación permite al usuario aceptar respuestas que serán ejecutadas y devolverán un valor. observa el ejemplo:

```
Prompter prompt: '¿Realizo el calculo del factorial?'
        defaultExpression: '10 factorial + 1'
```

esto implica el resultado de 3628801. También puedes sustituir la expresión "10 factorial + 1" por un espacio en blanco. Si haces esto, el prompter te pedirá la expresión. Tecléala y observa que trabaja de la misma manera.

Vidrios simples con interacción.

Afortunadamente, existen otras ventanas en Smalltalk/V que no son tan restrictivas como los prompters. Estas ventanas están dirigidas al usuario y bajo control en la programación al desarrollar aplicaciones. En forma básica Smalltalk/V proporciona clases definidas muy simples que muestran ventanas para que el usuario las incorpore fácilmente a sus aplicaciones propias. Además también cuenta con clases que definen ventanas muy complejas que dan base al bien desarrollado ambiente propio de Smalltalk/V.

Ventana y Vidrio de texto

Empecemos con un ejemplo muy sencillo que usa una ventana con un simple textPane. El código esta mostrado a continuación:

```
LearnWindow :=
    TextWindow
    windowLabeled: 'Estado de aprendizaje'
    frame: ((Display boundingBox leftTop rightAndDown: 100 @ 100)
        extentFromLeftTop: 400 @ 200).
```

Responde si (yes) al diálogo para definir **LearnWindow** como una variable global.

TextWindow es una variable de instancia de la clase **Window**, que cuando trabaja en combinación con un **TextPane**, provee la capacidad de editar texto. Enviando el mensaje **windowLabeled:frame** para crear una ventana que ocupa un área con una extensión de 400 @ 200 sobre la pantalla, con un título especifico centrado sobre la barra de títulos. Las hojas nuevas de trabajo (**workspace**) también son instancias de **TextWindow**.

Aunque esta nueva ventana conoce automáticamente como editar el texto interactivo por via del teclado del usuario o por la entrada del mouse, también se puede escribir texto desde la ventana Transcript estando activa o desde la ventana del workspace dentro del texto del vidrio (pane) de esta nueva ventana evaluando la siguiente expresión :

```
LearnWindow
    nextPutAll: 'I have learned everything about Smalltalk.';
    cr.
```

LearnWindow se usa aqui como un flujo (**Stream**), que ya se ha estudiado en algún capítulo anterior. Como tú sabes el mensaje **nextPutAll:** añade este argumento cadena al final de los contenidos del receptor. El mensaje cr entonces añade una línea comida.
Haz click en la nueva ventana para seleccionarla, activa la ventana y edita el texto dentro del **TextPane** de la ventana.
Para recuperar el contenido de esta ventana de otra ventana, activa una ventana diferente y evalúa la expresión con el **Show it**:

```
LearnWindow contents
```

Para cerrar la ventana, selecciona close en el sistema de menú de LearnWindow. O si estas en otra ventana, simplemente evalúa la expresión siguiente:

```
LearnWindow close.
```

Otro tipo de ventana simple son las cajas de mensajes (**Messages boxes**), que suelen ser usadas para desplegar el estado simple de la información o para mostrar mensajes de advertencia. Una vez que aparece una caja con un mensaje, el usuario debe hacer click con el mouse sobre el OK para que la caja desaparezca. Alternativamente, las cajas de mensaje pueden usarse para solicitar una respuesta falsa o verdadera. En este caso el usuario debe hacer click sobre el botón de si (yes), para una respuesta verdadera, o sobre el botón no (no) para una respuesta falsa. Alternativamente puede dar "enter" para activar el botón si (yes). Evalúa los siguientes ejemplos :

MessageBox message: ' no puedes hacer esto'

MessageBox confirm: ' continuo el proceso'

Menús

Otra forma de operar con la interacción en Smalltalk/V son los llamados menús. Los menús capacitan al usuario para crear una lista de campos para seleccionar alguna opción deseada. Un menú debe contener las características siguientes :

- Una lista de campos o entradas, a través de un arreglo de caracteres o simbolos.
- Barras divisoras de los campos, a través de un arreglo de indices.
- Una lista de respuestas, a través de un arreglo de simbolos de respuestas o cadenas correspondientes con los campos de entrada. Para aclarar esta definición de menú, evalúa el siguiente ejemplo

```
(Menu
        labelArray: #(Red Green Blue Yellow Black White)
        lines: #(3 5)
        selectors: #(red gren blue orange yellow black white))
                popUp
```

Apéndice I Glosario de Términos.

Ambiente de Programación.- Una facilidad que provee tres capacidades: una interfaz entre un programador orientado al lenguaje tales como Smalltalk y un lenguaje máquina especifico; herramientas que soportan el proceso de programación en el programador orientado a el lenguaje; y una librería de instrucciones preprogramada que pueden ser usadas para construir comandos mucho más elaborados.

Behavior (Comportamiento).- Lo que un Objeto puede hacer; el conjunto de mensajes que este responde; las operaciones (y su semántica) asociadas con un Objeto.

Bloque.- Una secuencia de cero o más expresiones separadas por un punto "." y rodeadas por paréntesis cuadrados. Los bloques son a menudo usados en estructuras de control.

Clase.- Una clase es un cuadro mediante el cual los objetos son definidos. Un conjunto de objetos similares son descritos e instanciados por su clase. Los objetos difieren sólo en los valores de los datos privados, llamados variables instancia. Una clase tiene un nombre, variables de clase, variables de instancia indexadas o nombradas, y un conjunto de métodos. Las variables de clase son variables nombradas que son compartidas por todas las instancias de la clase. Las variables de instancia son privadas, son datos locales que sirven para usar instancias de la clase.

Clase Abstracta.- Una clase que no tiene instancias o que podría pero no es intentado, porque las instancias no serían útiles. Sin embargo las instancias útiles son obtenidas de subclases de las clases abstractas.

Class Hierarchy Browser.- Una ventana usada para ver la biblioteca de clases (métodos y clases) y para hacer adiciones o modificaciones a esta biblioteca.

Concatenación.- Es una operación que se indica con el signo "," que permite un nuevo string o array (usualmente más larga) para ser construida de dos más cortas; por ejemplo, 'yester', 'day' => 'yesterday'.

Depurador.- Una herramienta que permite a el usuario rastrear la ejecución de un programa usando **Hop** y **Skip**, investiga los valores de las variables durante la ejecución, e inspecciona aquellos valores.

Diccionario.- Un contenedor que te permite correlacionar o asociar un Objeto con otro, un contenedor de pares de llaves-valores donde las llaves son los Objetos que sirven como indicadores para la información de interés y los valores son las datos relevantes.

Do It.- (Se encuentra en el menú Smalltalk) Evalúa el texto seleccionado en ese momento pero no imprime nada.

Enviando un Mensaje.- Solicitar que una simple expresión Smalltalk consistente de un receptor, un selector, y parámetros sean evaluados.

Estructura de Control.- Un mensaje que permite que la ruta de ejecución sea controlada.

Herencia.- El proceso que permite que los métodos y la representación provista por una superclase pueda ser usada automáticamente por una subclase.

Hop.- Brinca dentro de un método a el siguiente mensaje para ser evaluado; es decir, comienza la evaluación de el siguiente mensaje mediante la creación de una nueva entrada en la parte alta de el depurador y desplegando el comienzo de el método asociado con el siguiente mensaje.

Imagen.- El archivo que mantiene el rastro de el ambiente de programación como existió este en el momento en que salvaste la imagen.

Inspect.- (Se encuentra en el menú Inspect de un Inspector) Evalúa el texto seleccionado en ese momento y abre un inspector sobre el resultado.

Inspect It.- (En el menú Smalltalk) Evalúa el texto actualmente seleccionado y abre un inspector sobre el resultado.

Instancia.- Un miembro de una o más clases; por ejemplo, 5 es una instancia de las clases de Smalltalk, **Integer**, **Number**, **Magnitude**, y **Objeto**.

Jerarquía.- Una organización de elementos para el propósito de destacar las relaciones entre esos mismos elementos; típicamente ilustrado con una gráfica.

Jerarquía de Clases.- Las clases son organizadas jerárquicamente. Una clase pasa hacia abajo su estructura a todos sus descendientes. La raíz de clases provee la estructura necesaria para todas las clases.

Jump.- Causa que el depurador desaparezca y la ejecución continué desde el actual punto de ejecución.

Mensaje.- El mensaje es la estructura de control básica en los programas orientados a objetos. Esta es la manera por la cual los programadores se comunican con los objetos y los objetos se comunican con cualquier otro objeto. Si el método es similar a la especificación de un procedimiento en lenguajes tradicionales, el mensaje es similar a la invocación. El tipo de invocación a éste la mayoría de las veces

se ve como una llamada a una función. Es la combinación de el selector y parámetros; es decir, "**class**" en "25 **class**", "+2" en "1+2", y "**between:** 1 **and:** 5" en "3 **between:** 1 **and:** 5".

Mensajes en Cascada.- Múltiples mensajes a el mismo receptor. Técnicamente, cada mensaje es separado por un punto y coma y el receptor es especificado sólo en el primer mensaje; por ejemplo, "Transcript **space**; **space**; cr".

Mensaje Binario.- Un mensaje (selector) con un parámetro donde el selector consiste de caracteres no alfanuméricos; es decir, "+2" ("+") en "1+2".

Mensaje Keyworded.- Un mensaje (selector) con dos o más parámetros donde el selector consiste de keywords; es decir, mensaje "**between:** 1 **and:** 5" (selector "**between:and:**") en "3 **between:** 1 **and:** 5".

Mensaje Polimórfico.- Un mensaje con diversas implementaciones en diferentes clases; por ejemplo, "*" es polimórfico porque es entendido por diferentes tipos de objetos tales como fracciones y números de punto flotante.

Mensaje Unitario.- Un mensaje (selector) con ningún parámetro; por ejemplo, "**class**" en "5 **class**" es tanto un mensaje unitario como un selector unitario.

Método.- Un método es un procedimiento definido como parte de la descripción de una clase. Los métodos de una clase especifican como sus objetos -instancias de la clase- responderán a cualquier mensaje particular. El selector de mensaje compara el nombre dado en el encabezado de el método. Así el método es codificado dentro de lista de la descripción de la clase, esto explica como diferentes objetos pueden responder a el mismo mensaje de diferente forma. Esto es llamado polimorfismo también definido en esta sección.

Método Privado.- Un método para sólo ser usado dentro de la clase en la que éste es definido; no se crea para el uso de usuarios externos.

Método Público.- Un método que usuarios externos pueden utilizar.

Metodología.- Una serie claramente especifica de pasos y un orden parcial de esos pasos que lograrán un bien definido objetivo, es decir, la metodología de refinamiento iterativo.

Nombre de Ruta (Path).- Un nombre tal como "c:\v20\book\vw.exe" que incluye el drive y una ruta de directorio completa.

Objetos.- El objeto es una extensión de el tipo de datos abstractos. Estos encapsulan datos y estructuras de programa. Un objeto contiene datos privados junto con piezas de código llamados métodos que habilitan al objeto a responder apropiadamente a los mensajes. Un mensaje debe pedir que un objeto actúe sobre sí mismo, por ejemplo, para cambiar su estado, o actuar sobre otros objetos mediante el paso de futuros mensajes.

Operador.- Es una porción de expresión que especifica la acción que será realizada; por ejemplo, "+" en "1+2".

Operando.- Esa porción de expresión consistente de datos para el operador; por ejemplo, "1" y "2" en "1+2".

Parámetros.- Esa parte de el mensaje que excluye el selector; por ejemplo, no hay parámetros en "25 **class**", un parámetro "2" en "1+2", y 2 parámetros "1" y "5" en "3 **between:** 1 **and:** 5".

Pen.- Un Objeto que puede dibujar de un punto a otro sobre la pantalla. Este mantiene un nib, una localización, y una dirección. Una pluma puede ser colocado en un punto arbitrario, hacer un **goto** a un punto especifico, o hacer **go** hacia adelante una distancia arbitraria en cualquier dirección en la cual este apuntando (en las últimas dos situaciones, una línea es dibujada de la actual posición a el punto de destino si el estado del **nib** es **down**).

Pixel.- Una unidad de distancia, ya sea en la dirección de las X o de las Y, sobre la pantalla.

Polimorfismo.- Cuando el mismo mensaje es enviado a diferentes objetos, cada uno puede responder en diferente modo porque este tiene su propio método con el cual actuar sobre la petición. En lenguajes tradicionales, este es llamado operador sobrecargado y debe ser implementado mediante extenso chequeo de tipos de datos. En los lenguajes orientados a objetos, una facilidad extensa de tipificación no es necesaria porque cada objeto es el maestro de sus propias operaciones. Por ejemplo, el mensaje **close** puede ser enviado a un objeto archivo (file) o a un objeto ventana (window) con una diferente acción resultante cada caso. Como otro ejemplo, si el operador "+" es enviado a un objeto número o a un objeto string, el resultado sería diferente en cada caso; el mensaje es válido si existe un correspondiente método dentro de la especificación de la clase de número o de la de string.

Prompter.- Una ventana temporal pidiendo una respuesta textual.

Punto.- Un par de coordenadas X y Y; es decir, (10 @ 20) es un punto con la coordenada X igual a 10 y una coordenada Y igual 20.

Receptor.- El que recibe un mensaje.

Reusabilidad.- La noción de que algo es usado para otras cosas paralelas para las que no fue específicamente creado o diseñado.

Show It.- (En el menú Smalltalk) Evalúa el texto seleccionado y despliega el resultado.

Subclase.- Una clase que hereda de otra inmediatamente arriba de ésta en la jerarquía de clases.

Selector.- Esa parte de el mensaje que excluye los parámetros; es decir, "**class**" en "25 **class**", "+" en "1+2", y "**between:** 1 **and:** 5" en "3 **between:** 1 **and:** 5".

Self.- Una pseudo-variable que denota a el receptor. Esta es asociado con un método copia especifico y no puede ser cambiado.

Skip.- Evalúa el texto actualmente señalado y posiciones subsocuentes el depurador en el mensaje inmediato siguiente.

Stream.- Una colección indexada con un indicador de posición. Diversas operaciones son proveídas de manera que permiten uno o más elementos de la actual posición hacia adelante para ser accesado o reemplazado y que simultáneamente mover el indicador de posición como efecto secundario.

Super.- Una pseudo-variable que denota el receptor pero causando que la búsqueda de un método comience en la superclase de el método que contiene "super". Este no puede ser cambiado.

Superclase.- Una clase arriba de una clase dada en una jerarquía de clases.

Transcript.- Una ventana workspace que no puede ser cerrado; un ventana de texto que puede ser usada por el programador o el sistema como un tablero de despliegue de texto.

Variable.- Un nombre que refiere a un objeto arbitrario. El objeto especifico que este refiere en cualquier momento puede ser cambiado. Para cada nombre, el primer caracter debe ser alfabético, y todos los otros pueden ser ya sea caracteres alfabéticos o dígitos. Cuando es evaluada, una variable retorna el objeto a el cual este es unido; si ninguna asignación es explícitamente ejecutada, la variable es asignada a nil.

Variable de Clase.- Una variable que puede ser accesada por una clase (y subclases) y cualquiera de sus instancias. Las variables de clases serán usadas preferentemente con respecto a las variables globales.

Variables Globales.- Una variable que puede ser accesada en cualquier lugar. Esta comienza con una letra mayúscula; por ejemplo, "Transcript", "Object". Una variable global es creada al usarla por primera vez y respondiendo **Yes** a el prompt que pide la confirmación de la creación de la variable global. Una variable global como "Junk" es removida de el ambiente Smalltalk mediante la ejecución de la siguiente sentencia:

"Smalltalk **removeKey:** #Junk".

Variable Instancia.- Una variable usada para referir a una parte de una instancia; por ejemplo, "**numerator**" y "**denominator**" son variables instancia de la clase **fraction**.

Variables Locales.- Las variables que existen sólo durante la ejecución; en el contexto de un método, ya sea variables declaradas en el método o los parámetros de los métodos.

Variable Temporal.- Una variable que es creada por un contexto especifico y que inmediatamente desaparece cuando ese contexto no existe más; por ejemplo, las variables declaradas especificamente para ejecución, una serie de expresiones en un Workspace, o variables declaradas en el cuerpo de un método.

Walkback.- Una ventana que aparece cuando un comando de programación contiene un error de ejecución.

Workspace.- Una ventana de texto que puede ser usada por el programador para cualquier computación arbitraria.

Apéndice 2 Sintaxis del Lenguaje de Programación Smalltalk/V

Modo en que se especifica la sintaxis.

La especificación formal de la sintaxis es presentada usando el Formulismo Backus Naur Extendido (EBNF, Extended Backus Naur Formalism). El EBNF es usado aquí para precisar y especificar concienzudamente la sintaxis.

Una especificación EBNF es una secuencia de reglas de sintaxis. La parte derecha de cada regla define la sintaxis en términos de otros nombres de reglas y símbolos obligatorios de el lenguaje. Los paréntesis, (y), muestran grupos de términos alternativos. La barra vertical |, separa términos alternativos. Los paréntesis cuadrados, [y], identifican expresiones opcionales. Las llaves, { y } identifican expresiones las cuales pueden ocurrir cero o más veces. Una cadena es escrita como una secuencia de caracteres encerrada por dobles comillas, ("") e identifica símbolos obligatorios en la definición de el lenguaje. Una identificador es una secuencia de letras y dígitos comenzando con una letra.

A continuación se presentan las reglas de sintaxis, que pueden comenzar y terminar en un solo renglón; o bien, terminar en el siguiente o siguientes renglones. Cada definición de sintaxis en particular termina cuando se encuentra un punto; además se agregaron sangrías a los renglones que forman parte o son una continuación de el renglón precedente. Los números que se encuentran colocados al inicio de cada línea no forman parte de la sintaxis, únicamente sirven como referencia para localizar o indicar en la lista alfabética de todos los diferentes términos del lenguaje, donde se encuentra su definición en particular o en que partes se aplican, dicha lista se encuentra más adelante.

```
1 método = patrón_de_Mensaje [número_Primitivo] [temporales]
2         serie_de_Expresiones.
3 patrón_de_Mensaje = selector_Unitario | selector_Binario 4          nombre_de_Variable  |  keyword
nombre_de_Variable | {keyword 5          nombre_de_Variable}.
6 número_Primitivo = "<""primitive:" número ">".
7 temporales = "|" {nombre_de_Variable} "|".
8 serie_de_Expresiones = {expresión "."} [ ["^ "] expresión].
9 expresión = {nombre_de_Variable} ":="
```

```
10      (primario | expresión_de_Mensaje {";" mensaje_en_Cascada}).
11 primario = nombre_de_Variable | literal | bloque |
12      "(" expresión ")".
13 expresión_de_Mensaje = expresión_Unitaria |expresión_Binaria |
14      expresión_Keyword.
15 mensaje_en_Cascada = mensaje_Unitario | mensaje_Binario |
16      mensaje_Keyword.
17 expresión_Unitaria = primario mensaje_Unitario
18      {mensaje_Unitario}.
19 expresión_Binaria = (expresión_Unitaria | primario ) 20  mensaje_Binario {mensaje_Binario}.
21 expresión_Keyword = (expresión_Binaria | primario) 22 mensaje_Keyword.
23 mensaje_Unitario = selector_Unitario.
24 mensaje_Binario = selector_Binario (expresión_Unitaria | 25      primario).
26 mensaje_Keyword = keyword (expresión_ Binaria | primario)
27      {keyword (expresión_Binaria | primario)}.
28 bloque = "["[{":"nombre_de_Variable}"|"] 29    serie_de_Expresiones"]".
30 keyword = identificador ":".
31 selector_Binario = "-" | caracter_Selector 32      {caracter_Selector}.
33 selector_Unitario = identificador.
34 literal = número | string | constante_Caracter |
35      constante_Símbolo | constante_Arreglo.
36 constante_Arreglo = "#" arreglo.
37 arreglo = "(" {número | string | simbolo | arreglo |
38      constante_Caracter ")".
39 número = [digitos "r" ] [ "-"] digitos_Grandes ["." 40      digitos_Grandes]          ["e"          ["-"]
digitos_Grandes].
41 string = "'" {character| " ' ' " | " " " |" - "|"_"} " ' ".
42 constante_Caracter = "$" (caracter |" ' "| " "" " | "-" |"  ").
43 constante_Símbolo = "=" símbolo.
44 símbolo = selector_Unitario | selector_Binario|
45      keyword {keyword}.
46 indentificador = letra {letra | dígito}.
```

```
47 caracter = caracter_Selector | letra | digito |
48        "[" | "]" | "{" | "}" | "(" | ")" | "^" | ";" | "$" |
49        "#" | ":".
50 caracter_Selector = "," | "+" | "/" | "\" | "*" | "-" |
51        "<" | ">" | "=" | "@" | "%" | "|" | "&" | "?" | "!".
52 letra = letra_Mayúscula |
53        "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
54        "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
55        "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z".
56 letra_Mayúscula =
57        "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
58        "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
59        "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z".
60 dígitos = digito {digito}.
61 digito = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
62        "8" | "9".
63 dígitos_Grandes = digito_Grande {digito_Grande}.
64 digito_Grande = digito | letra_Mayúscula.
65 comentario = " " " {caracter | " ' " | " - " | " _ " } " " ".
66 nombre_de_Variable = identificador.
```

En la siguiente tabla se localizan en orden alfabético todos los términos involucrados en las definiciones anteriores. Los números corresponden a la línea en la cual se encuentra dicho término, y en el caso de que el número presente un signo menos (-) antes de él, significa que ahí se encuentra su definición.

arreglo		36	-37	37		
constante_Arreglo		35	-36			
digito_Grande		63	63	-64		
dígitos_Grandes	39	40	40	-63		
expresión_Binaria		13	-19	21	26	27
mensaje_Binario	15	20	20	-24		
selector_Binario	3	24	-31	44		
bloque		11	-28			
letra_Mayúscula	52	-56	64			

mensaje_en_Cascada		10	-15						
caracter		41	42	-47	65				
constante_Caracter		34	38	-42					
comentario		-65							
digito		46	47	60	60	-61	64		
digitos		39	-60						
expresión			8	8	-9	12			
serie_de_Expresiones	2	-8	29						
identificador		30	33	-46	66				
keyword			4	4	26	27	-30	45	45
expresión_Keyword		14	-21						
mensaje_Keyword		16	22	-26					
letra		46	46	47	-52				
literal		11	-34						
expresión_de_Mensaje	10	-13							
patrón_de_Mensaje		1	-3						
método		-1							
número		6	34	37	-39				
primario			10	-11	17	19	21	25	26
27									
número_Primitivo		1	-6						
caracter_Selector		31	32	47	-50				
string		34	37	-41					
simbolo		37	43	-44					
constante_Simbolo		35	-43						
temporales		1	-7						
expresión_Unitaria		13	-17	19	24				
mensaje_Unitario		15	17	18	-23				
selector_Unitario		3	23	-33	44				
nombre_de_Variable		4	4	5	7	9	11	28	-66

Referencia Bibliográfica.

Para profundizar más en el aprendizaje tanto de las propiedades de Smalltalk/V, como en su historia y diversas técnicas de programación más avanzadas, en seguida se proporciona una pequeña bibliografía que aunque no es muy amplia, si es muy útil cada uno de los títulos.

- Programación Orientada a Objetos.
  Aplicaciones con Smalltalk
  Autores: Angel Morales Lozano
  Franciscio J. Segovia Pérez

- Smalltalk Programming for Windows.
  Autores: Dan Shafer with
  Scott Herndon and
  Laurence Rozier.

- Object-Oriented Programming An Introduction
  Autor: Greg Voss

- Comparative Programming Languages
  Autor: Linda Weiser Friedman

- Análisis y Diseño Orientado a Objetos
  Autores: James Martin
  James J. Odell

- Object-Oriented Analysis and Design With Applications
  Autor: Grady Booch

- Discovering Smalltalk
  Autor: Wilf Lalonde