UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

U.A.C.P. y P. del C.C.H.
I.I.M.A.S.

"TUTORIALES EN LA RED INTERNET UTILIZANDO HTML, JAVASCRIPT Y JAVA"

TESIS

QUE PARA OBTENER EL GRADO DE :

MAESTRO EN CIENCIAS DE LA COMPUTACION

PRESENTA :

JESÚS ANTONIO CASTRO

MEXICO, D.F. **ENERO DE 1998**

Agradecimientos.

En los últimos tres años he aprendido que es difícil obtener logros en el individualismo. El apoyo y la colaboración de otras personas me ha permitido crecer a un ritmo más acelerado que antes. En este espacio quiero dejar patente mi sincero y profundo agradecimiento a quienes favorecieron ese crecimiento. En primer lugar a la Doctora Hanna Oktaba, quien revisó con mucho profesionalismo cada una de las propuestas de este trabajo, proporcionándome la guía necesaria para moldear un criterio general y para ubicar cada uno de los conceptos. Para mí, ha sido la luz al final del túnel. Agradezco a la Mat. Ana Luisa Solís González Cosío por aceptar dirigirme, a pesar de su ajustado calendario de trabajo. Mi especial gratitud a los Maestros Guadalupe Ibarguengoitia González, Marlene Peralta García y Antonio Hernández Sánchez por sus valiosos comentarios durante la revisión de esta tesis.

A mis profesores de la maestría, que me ayudaron no solo a obtener conocimientos, sino también los criterios relacionados con cada una de sus áreas de especialidad. A mis compañeros de estudios por la solidaridad y la fuerza de carácter que manifestaron cada vez que nos sentíamos como náufragos en un mar de interrogantes ante la cercanía de una fecha de entrega de proyecto.

Este trabajo va dedicado a mis seres queridos : Magali, Iliana, Marco Antonio y Erín quienes siempre me han apoyado para alcanzar mis metas.

La Paz, B.C.S. Diciembre de 1997.

Prefacio.

En nuestro trabajo cotidiano como profesores del Instituto Tecnológico de La Paz, tradicionalmente hemos tenido que enfrentar grandes problemas para conseguir información actualizada acerca de casi cualquier tema de estudio.

Ante la carencia de material bibliográfico, en muchos casos los alumnos han llegado a depender, casi totalmente, de las notas preparadas por los profesores. Los profesores, por su parte, se han enfrentado al problema de no contar con el material necesario y oportuno para mantener actualizadas esas notas.

Con el advenimiento y el rápido crecimiento de la red Internet, se avizora una solución para este problema de escasez de información, ya que la distribución de ésta puede hacerse casi tan rápido como se actualice. Esto es posible aún en el caso de organizaciones cuyos integrantes se encuentren dispersos en diferentes localidades.

Por medio de Internet, los esfuerzos que han realizado los profesores, de manera aislada en muchos de los institutos tecnológicos, podrán conjuntarse de manera organizada transformando en un tutorial las notas existentes para cada materia. De esta forma, se podrá contar con un tutorial para cada materia perteneciente a la retícula del plan de estudios más reciente de cada carrera profesional ofrecida por el sistema nacional.

Como una acción inmediata de implementación, registramos un programa de servicio social para los estudiantes y pasantes de las carreras de Licenciado en Informática y de Ingeniero en Sistemas Computacionales, por medio del cual cada prestador se compromete a elaborar un tutorial para una materia de cualquier carrera ; contando con la asesoría de un profesor de la materia (como experto del conocimiento) y con orientación de nuestra parte para la elaboración del tutorial.

El **Tutorial de Lenguaje C++** descrito en este trabajo es el primer resultado de ese programa de servicio social, y en él se utilizaron las notas que elaboramos para una materia optativa de la carrera de Licenciado en Informática y la de Ingeniero en Sistemas Computacionales.

Agradecemos el apoyo de la Universidad Autónoma de Baja California Sur, que a través del Departamento de Sistemas Computacionales y de Protel ha posibilitado la prueba y publicación en Internet del **Tutorial de Lenguaje C++** , el cual puede visitarse en:

http ://www.uabcs.mx/teclapaz/tutorc/index.htm

INDICE

Introducción.

El presente trabajo propone una metodología para producir tutoriales que puedan manejarse a través de la red Internet, seleccionando algunas de las principales herramientas disponibles en la actualidad (1997) para la elaboración de páginas Web.

El objetivo es promover la generación de aplicaciones que cubran los contenidos de materias correspondientes a planes de estudio de carreras de nivel superior, factibles de utilizarse de manera aislada en computadoras personales o a través de la red Internet.

A continuación se presenta una breve descripción y la manera en que está distribuido el material contenido en el presente trabajo.

En el **Capítulo 1.- Sistemas Tutoriales** , se describen algunos conceptos básicos relacionados con el tratamiento del conocimiento por medio de tutoriales que utilizan la computadora como herramienta principal.

El **Capítulo 2.- Sistemas tutoriales en Internet** , describe conceptos básicos acerca de la red Internet y plantea las ventajas de utilizarla para el manejo de tutoriales pertenecientes a materias de estudios escolarizados.

En los Capítulos 3, 4 , y 5 se describen brevemente algunas herramientas para el desarrollo de tutoriales, aprovechando los servicios de la WWW (World Wide Web) como son los lenguajes HTML, JavaScript, y Java, ya que son los que actualmente se adaptan mejor para el manejo de los servicios de Internet en un sistema interactivo y de múltiples ventanas. Los sistemas construidos con estos lenguajes tendrán capacidad para ejecutarse por medio de un visualizador (browser, tal como el NetScape Navigator) , instalado en una red o en una computadora aislada (por ejemplo una PC).

En el **Capítulo 3**.- **Tutoriales básicos**, se empieza por definir los tutoriales básicos y se termina con una descripción del lenguaje HTML. Ahí se definen las presentaciones Web, su organización y sus páginas. En ese capítulo se mencionan también algunos de los visualizadores más utilizados para el despliegue de páginas. La mayor parte del capítulo se dedica a describir algunas de las características básicas de HTML, sin pretender ser una referencia completa.

En el **Capítulo 4.- Tutoriales y JavaScript**, se describen algunas de las características de JavaScript, enfatizando los aspectos del lenguaje que se requieren en los tutoriales y que no son cubiertos por HTML. Al inicio del capítulo, se definen algunos conceptos básicos de la Programación Orientada a Objetos, continuando con una breve referencia al lenguaje JavaScript.

El **Capítulo 5.- Tutoriales mejorados con Java**, tiene una organización similar al Capítulo 4. Se describen al principio las características básicas del lenguaje de programación Java y se ejemplifica en aquellas que considero con más posibilidades de utilización en los tutoriales. El capítulo termina con un sencillo ejemplo de animación.

En el **Capítulo 6.- Metodología propuesta**, se presenta una serie de criterios que espero sirvan de apoyo a quienes se interesen en la elaboración de tutoriales del tipo CAI. Los ejemplos para este capítulo se tomaron del **Tutorial de Lenguaje C++**, el cual está dirigido a personas que posean un nivel de conocimiento de principiante a intermedio acerca de dicho lenguaje.

Capítulo 1. Sistemas tutoriales.

Congruentes con la gran disposición que, como humanos, tenemos hacia la clasificación o catalogación de los entes de aprendizaje (los objetos y los conceptos con los que interactuamos durante nuestra existencia), ubicaremos el presente trabajo dentro de los llamados *sistemas basados en el conocimiento*, los cuales a su vez pertenecen a una categoría llamada *inteligencia artificial*. Dada la naturaleza del tema, deberíamos hablar también de la Sicología, la Lingüística, la Pedagogía, la Sociología, entre otras ciencias , pero los alcances de este trabajo no corresponden a un estudio multidisciplinario de tal magnitud.

En lo que sigue describiremos brevemente algunas de las áreas de conocimiento relacionadas con la creación de sistemas tutoriales.

1.1.- Inteligencia artificial.

La inteligencia artificial es un campo de estudio que busca explicar y emular inteligencia, desarrollándola en términos de procesos computacionales que, si son utilizados correctamente por un programa, éste puede exhibir un comportamiento inteligente[LAURA92]. Esto es, la inteligencia artificial expresa la pretensión de modelar, por medio de máquinas computadoras, la inteligencia del ser humano. Esta pretensión se basa en la experiencia que la humanidad ha acumulado en el campo de la elaboración de modelos para interpretar la naturaleza y el comportamiento de la fracción del Universo con la cual tiene contacto.

Para poder modelar un sistema, el creador del modelo deberá contar con herramientas útiles para representar tanto la forma como el comportamiento del sistema, de tal manera que otros seres humanos puedan comprender el sistema a través del modelo creado.

La dificultad para precisar una definición de inteligencia artificial radica en que no se conoce con certeza qué es y cómo funciona la inteligencia. Por esta misma razón, es difícil construir un modelo que la represente con una buena aproximación. Sea lo que fuere la inteligencia, es un hecho aceptado que los seres humanos utilizan la inteligencia para aprender, y que el centro de la actividad intelectual se da en el cerebro.

1.2.- Aprendizaje.

Un individuo aprende en la medida que es capaz de almacenar y utilizar cierta cantidad de conocimiento para utilizarlo posteriormente. Cuando un individuo aprende, sufre un cambio conductual que lo habilita para realizar nuevas acciones con base en el conocimiento adquirido.

Si aceptamos este enfoque, podemos afirmar que un individuo ha aprendido cuando es capaz de resolver un problema que no podía resolver antes de adquirir conocimiento nuevo.

Aunque no podemos conocer con precisión los mecanismos que se generan durante el aprendizaje, sí podemos adecuar el ambiente en el cual se coloca el individuo para aprender.

Una de las herramientas más utilizadas en la actualidad como auxiliar del aprendizaje es la computadora, que estimula principalmente los sentidos de la vista y el oído. Por medio de esta herramienta es posible presentar una serie de imágenes y sonidos que mantengan el interés del aprendiz. Pero esto de poco serviría si no se le incluyen los conceptos propios del dominio de conocimiento en cuestión. Por otro lado, un individuo aprende mejor cuando centra su atención en pocos conceptos a la vez y cuando éstos están relacionados. También es conveniente que el aprendiz revise su grado de avance en la adquisición de conocimiento, esto es, que verifique su capacidad de resolver una parte del problema total, para que, apoyándose en el conocimiento adquirido, minimice el tiempo necesario para la adquisición del conocimiento que le falta para la solución del problema completo.

Todo esto nos lleva a la conclusión de que los sistemas auxiliares para el aprendizaje deben contar con :

- un dominio de conocimiento claramente definido,
- una interfaz del usuario que sea clara, atractiva y que facilite la interacción humano/máquina.

1.3.- Instrucción asistida por computadora.

Desde la década de los 50, el uso de las computadoras en la educación ha sido considerado como un campo importante de aplicaciones. El acrónimo CAI (Computer Aided Instruction , Instrucción Asistida por Computadora) fue introducido en esa época y desde entonces han sido desarrollados muchos programas CAI.

Los tradicionales programas CAI pueden considerarse como los descendientes evolutivos de los libros, ya que al igual que ellos, están organizados estáticamente de tal forma que contienen tanto el dominio de conocimiento como el conocimiento tutorial de los maestros, como expertos humanos. En los libros se encuentran algunas ayudas, tales como: secciones, capítulos, índices, tablas y figuras; son las herramientas con las que se facilita la presentación de un tema, y los autores tienen conocimiento tanto en el dominio de lo que hay que comunicar como en la escritura del propio libro.

Un libro puede propiciar varios niveles de lectura, proporcionar referencias cruzadas entre secciones y suministrar glosarios adecuados. No obstante esto, el autor tiene que dar

todo por adelantado. Un libro no puede responder a preguntas inesperadas del lector. Tampoco puede modificarse "al vuelo" el conocimiento presentado, de tal forma que se adapte a un lector específico. Las limitaciones propias del medio de impresión no permiten mucha flexibilidad y dinamismo en el acceso al conocimiento contenido en el libro.

Los autores de programas CAI hacen los mismo: de antemano piensan en acciones educacionales, anticipan las circunstancias que requieren decisiones y escriben el código apropiado que permita capturar tales decisiones. Por consiguiente, los programas CAI tradicionales aprovechan la experiencia tutorial de los maestros expertos y directamente reflejan esta habilidad en el comportamiento de los programas. Esta circunstancia hace poderoso el enfoque de los sistemas CAI, pero, al mismo tiempo, es su falla mas importante. De hecho, pocos maestros pueden anticipar *todos* los errores de concepto que un estudiante puede adquirir. Lo que es peor, es prácticamente imposible realizar programas de aplicaciones que contengan *todas* las decisiones imaginables. Aún cuando es verdad que los programas CAI, una vez que son desarrollados y probados pueden ser usados por un gran número de personas, también es cierto que son difíciles de modificar.

Podemos clasificar los programas CAI de acuerdo a su "orientación", es decir, respecto al papel que juega la computadora en el control de la actividad de aprendizaje. En los dos extremos están, por una parte, el ambiente de aprendizaje (por ejemplo, los micro mundos de LOGO en los que el estudiante puede "aprender al estar descubriendo"); en el otro extremo, existen aplicaciones en las que el estudiante tiene que obedecer una estrategia de aprendizaje estricta que le permita mejorar la calificación en la evaluación del aprendizaje.

Hay que hacer notar que no se puede precisar una línea divisoria entre los diferentes tipos de programas. En realidad, los programas CAI pueden verse como un continuo desde un tutoreo con estrategias estrictamente rígidas hasta el tutoreo con una autonomía completa.

Las principales desventajas de los programas CAI se pueden resumir de la siguiente manera:

(a).- La calidad de los programas de aplicaciones está estrechamente relacionada, en muchos sistemas de tutoreo dirigido, con la habilidad del autor para anticipar (e incluir en ellos) tantas respuestas del estudiante como sea posible y entonces poder especificar la trayectoria de tutoreo que resulte mas apropiada.

(b).- Una estrategia de enseñanza, trasladada a lineas de código, no se ajusta a las necesidades específicas del estudiante.

(c).- La autonomía que ofrecen los sistemas de tutoreo menos dirigido representan un obstáculo para aquellos estudiantes que no tienen la posibilidad de aprovechar esta autonomía.

(d).- Los programas CAI, a menudo, son desventajosos en lo referente al costo, dado que desarrollar y mantener tales programas implica invertir recursos de considerable magnitud.

No obstante esto, un juicio objetivo de los programas CAI debe tomar en cuenta que su meta es muy interesante, además de que este campo es inherentemente interdisciplinario. Las disciplinas cuyo impacto ha resultado más significativo en el desarrollo de los sistemas CAI son : sicología, ciencia cognitiva, pedagogía, epistemología, interacción hombre-máquina y lingüística.

Se puede anticipar con seguridad que éstas áreas se acercarán cada vez más, debido a que comparten un interés común en los procesos de comunicación del conocimiento.

1.4.- Sistemas expertos.

Los *sistemas expertos* son una familia de sistemas que pertenecen a la inteligencia artificial, y su interés principal con relación a propósitos educativos reside en la circunstancia, tanto histórica como conceptual, de que constituyen un enlace natural entre la inteligencia artificial y los sistemas tutoriales [FISHEE93].

Un sistema experto es un sistema basado en la computadora que es capaz de resolver problemas complejos en dominios específicos, mostrando un nivel de desempeño comparado con el de los expertos humanos.

La dificultad principal reside en el problema de representar el conocimiento del experto. Además del dominio de conocimiento especifico, las estrategias para la resolución de los problemas del experto también deben capturarse y codificarse de manera conveniente. Es posible que el experto use estas habilidades a un nivel inconsciente, por lo que el proceso de formalizar el conocimiento (hacer patente un dominio) es un objetivo que constituye un reto y que requiere la intervención del *ingeniero del conocimiento* para poder codificar el conocimiento del experto.

Como se muestra en la Figura 1.1, la estructura básica de un sistema experto consta de : una *base del conocimiento* (que incluye *hechos* y *reglas* codificadas apropiadamente) , y una *maquina de inferencias*, cuyo cometido es el de activar las reglas para que se pueda obtener la solución del problema. Se han logrado desempeños excelentes por parte de algunos sistemas como : MYCIN (infecciones bacterianas), DENDRAL (química orgánica) y PROSPECTOR (en geología).

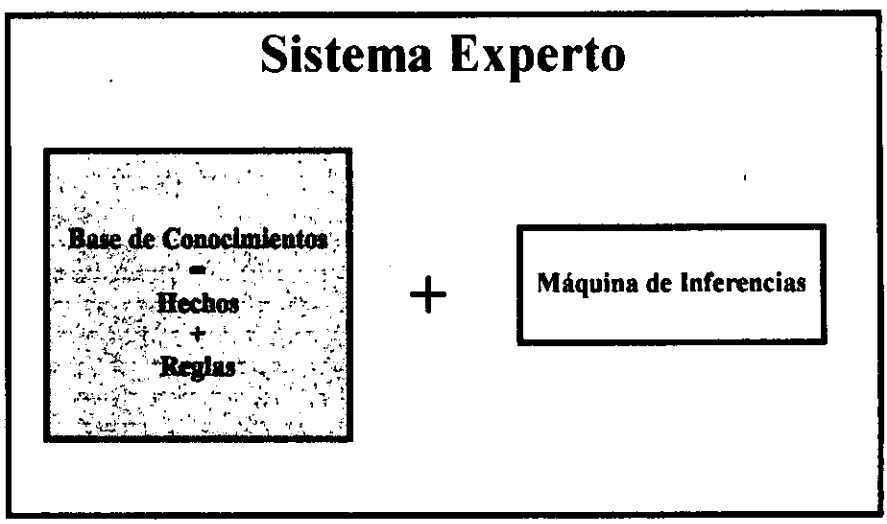


Figura 1.1.- Estructura básica de un sistema experto.

Es evidente que uno de los principales campos de aplicación de los sistemas expertos debe ser la educación.

De hecho, los sistemas expertos, además de proporcionar una solución a un problema, también muestran los pasos de "razonamiento" que le permitieron llegar a esa conclusión. Sin embargo, la aplicación de los sistemas expertos a los ambientes educativos muestra grandes dificultades, ya que el arte pedagógico es algo más que solo codificación y procesamiento del conocimiento. Un maestro, como experto humano, no solo muestra el camino hacia la solución correcta, sino que explora otras posibles vías de solución, y su meta principal es que el estudiante sea capaz de resolver toda una clase de problemas. Para lograr este objetivo, el maestro adapta toda su estrategia de enseñanza a las necesidades de un estudiante específico.

Es por esto que actualmente existe el consenso de que un sistema tutorial eficiente debe poseer, además del dominio del experto, estrategias de enseñanza apropiadas que se puedan adaptar a las necesidades específicas de cada estudiante. Finalmente, la interfaz de comunicación entre el sistema y el estudiante debe ser, al mismo tiempo, muy amigable y poderosa [FISHEE93].

1.5.- Sistemas tutoriales inteligentes.

El acrónimo ICAI (Intelligent Computer Aided Instruction , Instrucción Inteligente Asistida por Computadora) evolucionó a partir de CAI para denotar el aspecto de la investigación educativa en la inteligencia artificial. No obstante esto, en la actualidad "ICAI" ha sido reemplazado por el acrónimo "ITS" (Intelligent Tutoring System, Sistema Tutorial Inteligente) acuñado por Sleeman y Brown [SLEED82]. Aún cuando ambas designaciones se encuentran en uso, hay una ligera preferencia por "ITS".

Los *sistemas tutoriales inteligentes* surgieron en la década de los setenta como resultado de la combinación de técnicas de la *inteligencia artificial* y de los métodos clásicos de enseñanza. El objetivo de los sistemas tutoriales inteligentes es proporcionar una mayor flexibilidad a los tutoriales manejados por computadora y lograr que éstos permitan una mejor interacción con el usuario.

Para lograr este objetivo deberá dotarse a dichos sistemas con la capacidad de "razonar" y resolver problemas en su dominio de aplicación. Asimismo, cada sistema tutorial inteligente deberá mantener un modelo del conocimiento del usuario para poder actuar con mayor "sensibilidad" ante el comportamiento de éste. Por último, es indispensable desarrollar una interfaz amigable que posea la capacidad de "dialogar" en lenguaje natural.

Con los sistemas tutoriales inteligentes se pretende capturar el conocimiento de los expertos, crear interacciones con las instrucciones en forma dinámica , y asi poder tomar decisiones no previstas por los expertos.

La meta ideal de un sistema con capacidad tutorial completamente autónoma se encuentra en la fase de investigación.

Es deseable que los sistemas tutoriales inteligentes sean operativos y mantenibles, para poder aplicarlos en el ambiente educativo. Esto puede lograrse en la medida que el modelo de estos sistemas sea adaptado a las herramientas computacionales existentes y evitar de esta manera el desarrollo de sistemas que solo lleguen al nivel de prototipos.

En la Figura 1.2 se muestra la estructura de lo que debe ser un sistema tutorial inteligente.

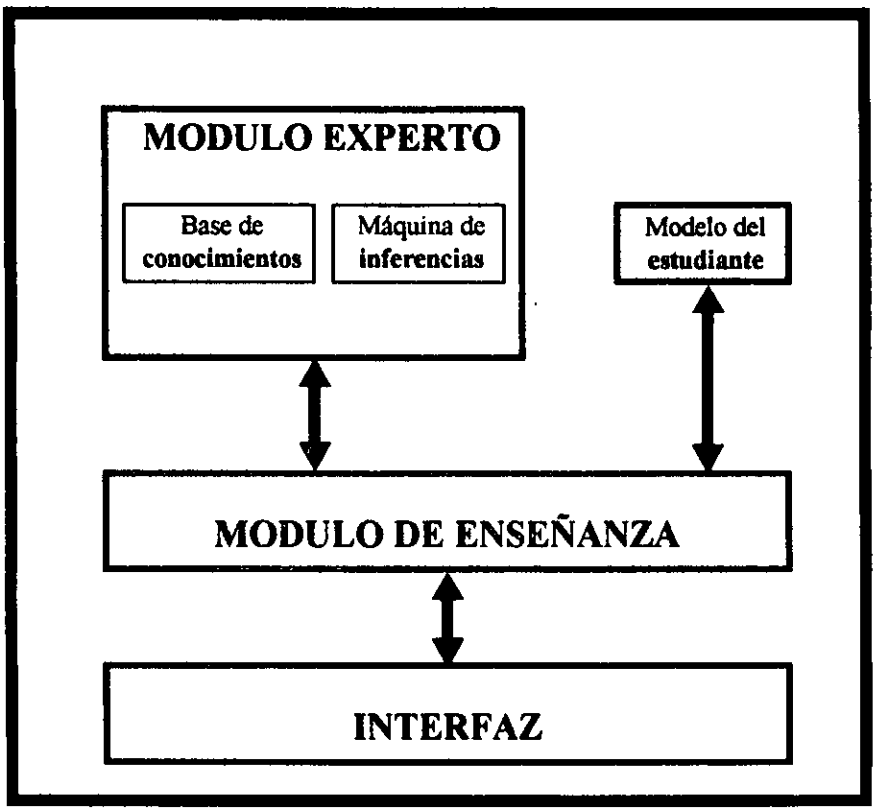


Figura 1.2.- Estructura de un sistema tutorial inteligente.

1.5.1.- Arquitectura de un sistema tutorial inteligente.

Los sistemas tutoriales inteligentes pueden considerarse compuestos de los cuatro módulos mostrados en la Figura 1.2 , los cuales se describen a continuación :

1.5.1.1.- Módulo experto

El módulo experto consiste de un sistema experto que contiene el conocimiento al cual desea acceder el estudiante. Mas bien es un modelo de los conocimientos y habilidades que han de ser enseñados.

En este módulo se cumplen las siguientes funciones:

- **Generación de preguntas, respuestas y descripciones** : en este sentido, el módulo actúa como fuente (base de conocimientos) del tema a enseñar.
- **Generación de soluciones y evaluaciones** : el módulo genera las soluciones posibles a los problemas planteados (máquina de inferencias) , dando una pauta para la evaluación del desempeño del estudiante.

1.5.1.2.- Modelo del estudiante.

El modelo del estudiante se refiere a quien utilizará al sistema tutorial inteligente para aprender acerca de un tema específico. El sistema deberá reconocer el comportamiento del estudiante a partir de modelos de patrones de conocimiento correcto o incorrecto. Generalmente, el conocimiento que posee el estudiante es un subconjunto del conocimiento que posee el experto. Cada tema de la base de conocimientos se asocia a una medida de la destreza obtenida por el estudiante en ese tema. Tal tipo de representación se conoce como *traslape*, debido a que se obtiene de la superposición del conocimiento poseído por el estudiante, y del conocimiento que se encuentra en el módulo experto. El modelo del estudiante debe incluir, además, una fase llamada *diagnóstico*, la cual deberá encargarse de medir el grado de conocimiento del estudiante con respecto a la base de conocimiento del sistema.

En la elaboración del modelo del estudiante se tienen que enfrentar los tres problemas siguientes:

1. El reconocimiento de errores y malas interpretaciones.
2. La interpretación del comportamiento del estudiante.
3. Respuestas no confiables.

1.5.1.3.- Módulo de enseñanza.

En este módulo, las estrategias de enseñanza, en lugar de estar empotradas en mecanismos de ramificación de un programa, deben derivarse de las interacciones con reglas especializadas que representan la experiencia pedagógica. La habilidad para comunicarse deberá almacenarse en forma de principios generales, de tal manera que el sistema pueda adaptar y mejorar las estrategias tutoriales a medida que transcurra el tiempo.

Las decisiones didácticas son complicadas y constituyen un problema que deberá atacarse con proyectos de largo plazo.

A continuación se desglosan los requisitos básicos que debe incluir un módulo de enseñanza ideal.

Módulo de enseñanza

Nivel Global : Planeación.

- Reconocimiento de submetas.
- Selección de tópicos.
- Elección de una secuencia de episodios de instrucción.
- Selección del nivel tutorial (elemental, avanzado, etc.).

Nivel Intermedio : Actividad individual.

- Selección de la siguiente actividad.
- Intervención en el proceso de aprendizaje.
- Suspensión de una actividad.

Nivel Local : Interacción.

- Selección del medio educativo.
- Explicaciones.
- Correcciones.
- Elección del lenguaje.
- Manejo de preguntas imprevistas.
- Guía para la ejecución de actividades.
- Selección de ejemplos.

En el cuadro anterior se reconocen tres niveles básicos en las estrategias de enseñanza. La lista parcial de actividades muestra la dificultad de la tarea pedagógica, ya que se carece de una especificación precisa de sus constituyentes básicos. Esto último hace que su tratamiento sea mas complejo que el de la mayoría de los dominios de conocimiento sobre los que se aplica.

Mas allá de los principios pedagógicos generales, en éste módulo se incluyen las estrategias de enseñanza directamente relacionadas con el dominio de conocimiento.

1.5.1.4.- Módulo de interfaz.

El objetivo del módulo de interfaz consiste en manipular la forma final de las decisiones didácticas. La importancia de la interfaz radica en que puede hacer que la presentación de un tema sea mas o menos comprensible, y afectar el nivel de aceptación que el estudiante tenga por el sistema. Esto hace que los avances tecnológicos en los equipos de cómputo lleguen a guiar el diseño de los sistemas.

Los sistemas tutoriales inteligentes que existen actualmente muestran varias habilidades que permiten manejar expresiones verbales, por medio del pseudo lenguaje natural.

1.6.- Delimitación del trabajo.

El presente trabajo no pretende realizar un análisis exhaustivo de las posibilidades que existen para la creación de todos los tipos de tutoriales mencionados en las secciones anteriores.

Nuestro interés se basa en la creación de tutoriales del tipo CAI descrito anteriormente y que puedan utilizarse de una manera práctica y eficiente por medio de la red Internet, y que cubra los contenidos de las materias que se cursan en las instituciones educativas de nivel superior de nuestro país, tomando en cuenta las limitaciones de los equipos y de las instalaciones con que se cuenta en la mayoría de ellas.

La decisión de utilizar HTML tiene como fundamento el hecho de que este lenguaje se basa en el protocolo HTTP ofrecido por la World Wide Web para administrar la transferencia de hipertexto entre los visualizadores y los servidores.

Para evitar que se sobrecargue de trabajo al servidor, es necesario utilizar lenguajes tales como JavaScript y Java, los cuales permiten que la mayor cantidad de procesamiento se realice en el extremo cliente de cada conexión, dejándole al servidor sólo aquellas tareas que se relacionan con la transferencia de archivos entre él y los clientes (además de las tareas relacionadas con la creación y la modificación de archivos en los dispositivos de almacenamiento del lado del servidor). Además, la utilización de estos lenguajes incrementa la seguridad en la red, puesto que restringen la grabación de información en el lado del cliente.

Capítulo 2.- Sistemas tutoriales en Internet.

En este capítulo se presentan los conceptos básicos que un elaborador de tutoriales debe conocer acerca de la red Internet, así como las principales razones por las que debe utilizarse ésta para el manejo de tutoriales asociados a materias que se cursan en estudios formales.

2.1.- Internet.

Internet es una red internacional formada por un conjunto de subredes de instituciones educativas, gubernamentales, empresariales , y comunidades de usuarios, que creció a partir de una red estadounidense llamada ARPAnet (Advanced Research Projects Agency net ; red de la Agencia de Proyectos de Investigación Avanzada). Las distintas redes que forman Internet se comunican entre sí a través de compuertas (también conocidas como enrutadores), utilizando el protocolo de comunicaciones TCP/IP (Transmission Control Protocol / Internet Protocol ; Protocolo de Control de Transmisión / Protocolo Internet). El término TCP/IP no es una entidad única que combina dos protocolos, sino el nombre genérico para una familia de protocolos que tienen comportamientos similares y que forman un conjunto más grande de programas que proporciona servicios de red como : registro remoto de entrada, transferencia remota de archivos y correo electrónico [PARKT95].

2.1.1.- Direcciones Internet.

Las direcciones de red le indican a un sistema dónde debe entregar un paquete de datos (datagrama), de una manera similar a como ocurre con las direcciones de correos.

Debe tenerse precaución al leer el término **dirección**, ya que a menudo los protocolos de comunicaciones lo utilizan para referirse a cosas distintas como : un destino, el puerto de una máquina, una localidad de memoria, una aplicación , etc.

Para el direccionamiento en Internet se utilizan comúnmente tres términos : nombre, dirección y ruta.

Un **nombre** es una identificación especifica de una máquina, un usuario o una aplicación. Por lo general, el nombre es único y proporciona un destino absoluto para el datagrama.

Una **dirección** identifica la localización del destino, por lo general su localización física o lógica en una red.

Una **ruta** le indica al sistema cómo hacer llegar un datagrama a una dirección especificada.

El nombre del destinatario se utiliza con frecuencia , ya sea por medio de la especificación de un nombre de usuario o un nombre de máquina. Un paquete de software de red, conocido como **servidor de nombres**, tratará de descifrar la dirección y la ruta a partir del nombre dado.

El uso del servidor de nombres tiene las siguientes ventajas :

1.- Hace que el direccionamiento y el enrutamiento resulten irrelevantes para el usuario.

2.- Le da al sistema o al administrador de la red una gran libertad para modificar la red según lo requiera, sin necesidad de informar a cada máquina de usuario.
Mientras una aplicación pueda tener acceso al servidor de un nombre , cualquier modificación en el enrutamiento podrá ser ignorada, tanto por la aplicación como por los usuarios.

2.1.2.- Direcciones IP.

TCP/IP utiliza una dirección de 32 bits para identificar una máquina y la red a la cual está conectada. Las direcciones IP identifican la conexión de la máquina a la red, no la máquina en sí ; por lo que, siempre que se modifique la localización de una máquina en la red, deberá modificarse la dirección IP. Esta dirección está formada por un conjunto de números separadas por puntos, como por ejemplo : 126.40.7.84

Unicamente el Centro de Información de Red (NIC, Net Information Center) puede asignar las direcciones IP o Internet . Existen cuatro clases de formatos para direcciones IP, cada una de las cuales se utiliza dependiendo del tamaño de la red, como se muestra en la Tabla 2.1.

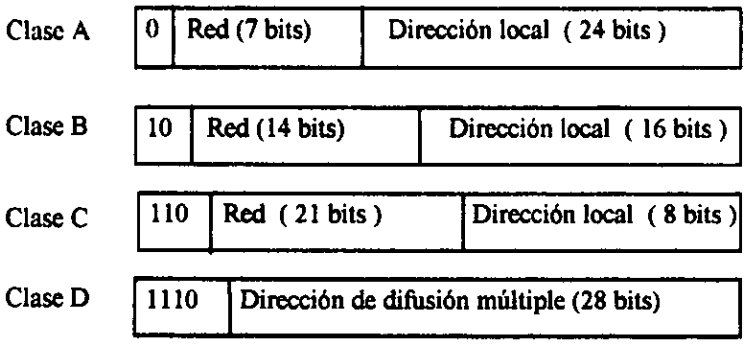


Tabla 2.1.- Estructuras de clase de dirección IP.

La clase se identifica mediante la primera secuencia de bits, por ejemplo 1 bit para la Clase A, 2 para la Clase B , etc.

La clase puede determinarse a partir de los 3 primeros bits de orden más alto. En la mayoría de los casos basta con los dos primeros bits, ya que existen pocas redes de Clase D.

Las direcciones de **Clase A** corresponden a **redes grandes con muchas máquinas**. En estos casos se necesitan los 24 bits para la dirección local (también conocida como dirección de anfitrión). La dirección de red se conserva en 7 bits, lo que limita el número de redes que se pueden identificar.

Las direcciones de **Clase B** sirven para redes de **tamaño intermedio**, con direcciones locales o de anfitrión de 16 bits y direcciones de red de 14 bits.

Las direcciones de **Clase C** tienen solo 8 bits para la dirección local o de anfitrión, limitando el número de dispositivos a **256** y dejando 21 bits para la dirección de red.

Por último, las direcciones de **Clase D** se utilizan con fines de **multidifusión**, cuando se requiere una difusión general a más de un dispositivo. Las longitudes de cada sección de las direcciones IP se han seleccionado cuidadosamente para proporcionar la máxima flexibilidad en la asignación de las direcciones local y de red.

Las direcciones IP están formadas por cuatro conjuntos de 8 bits. Por comodidad, a menudo estos bits se representan con sus equivalentes separados por un punto, por lo que el formato de dirección puede ser **red.local.local.local** para la Clase A hasta **red.red.red.local** para la Clase C. Las direcciones IP por lo general se escriben en sus equivalentes decimales, en lugar de escribir largas cadenas binarias. La forma decimal es ese familiar número de anfitrión que los usuarios de red están acostumbrados a ver, como por ejemplo : 145.12.14.25 ; en donde 145.12 es la dirección de red, y 14.25 es la dirección local o de anfitrión. Por supuesto que las direcciones reales están formadas por conjuntos de unos y ceros. El nombre apropiado para la notación decimal utilizada para las direcciones IP es : **notación cuadrática con punto** [PARKT95].

2.1.3.- Sistema de Nombre de Dominio.

En lugar de utilizar la dirección IP completa de 32 bits, muchos sistemas adoptan nombres más significativos para sus dispositivos y redes. Por lo general los nombres de las redes reflejan el nombre de la organización. Los nombres individuales de los dispositivos de la red pueden ser desde nombres descriptivos en redes pequeñas hasta convenciones más complejas de asignación de nombres en redes más grandes.

A fin de resolver el problema de los nombres de red, el Centro de Información de Red (NIC, Network Information Center) mantiene una lista de los nombres de red y de las direcciones correspondientes de las compuertas de red.). Cuando el número de redes se hizo demasiado grande, este sistema creció de una sencilla lista de archivo plano a un sistema más complejo conocido como Sistema de Nombre de Dominio (DNS, Domain Name System) [PARKT95].

2.1.3.1.- Estructura del Sistema de Dominios.

El DNS establece una forma de administración de nombres, distribuyendo en diferentes grupos la responsabilidad de subconjuntos de nombres. A cada nivel de este subsistema se le llama **dominio**. Los dominios se separan por puntos, como se muestra en los siguientes ejemplos :

balandra.uabcs.mx
ux.cso.uiuc.edu
servidor.dgsca.unam.mx
nic.ddn.mil

Al leer un nombre de izquierda a derecha, cada dominio será un subconjunto del que se encuentre a su derecha. Por ejemplo en el nombre **servidor.dgsca.unam.mx**, **servidor** es el nombre del equipo anfitrión (host, una computadora con una dirección IP). El nombre para esa computadora se asigna y se mantiene por el grupo **dgsca**, que es el lugar donde se localiza el equipo anfitrión. La dirección general **dgsca** es parte de la Universidad Nacional Autónoma de México (**unam**), que a su vez se encuentra localizada en el país México (**mx**). De esta manera, el dominio **mx** está compuesto de todas las computadoras anfitrión conectadas a Internet en México, el dominio **unam.mx** contiene a todas las computadoras anfitrión de la Universidad Nacional Autónoma de México conectadas a Internet, y así sucesivamente.

Cada grupo puede cambiar cualquier dominio que se encuentre dentro de él. Por ejemplo, si **unam** decide crear otro grupo denominado **egresados**, lo puede hacer sin solicitar ningún permiso ; solamente tiene que agregar el nuevo nombre a su parte de la base de datos mundial. Algún navegante de la red descubrirá después el nuevo nombre (**egresados.unam.mx**). De manera similar, **dgsca** puede adquirir una nueva computadora, asignarle un nombre y conectarla a la red sin pedir permiso.

Si cada grupo respeta las reglas y se asegura de que los nombres que asigna son únicos, ningún nombre en Internet estará repetido. Es posible que existan dos máquinas llamadas **paco**, pero solamente en dominios distintos por ejemplo :

paco.egresados.unam.mx
paco.cso.uiuc.edu

Cada organización debe establecer un procedimiento para solicitar, ante la persona responsable de la administración de cada nivel, la creación o la modificación de un nombre.

Los dominios de la jerarquía superior, tales como **edu**, fueron creados por consenso cuando se diseñó el Sistema de Nombre de Dominio (DNS). Originalmente, solamente existían en la jerarquía superior los seis dominios siguientes :

Dominio	Utilización
com	Organizaciones comerciales (negocios)
edu	Organizaciones educativas (universidades, tecnológicos, secundarias, etc.)
gov	Organizaciones gubernamentales (sin incluir a la milicia)
mil	La milicia (el ejército, la marina, etc.)
org	Otras organizaciones.
net	Recursos de la red.

Cuando Internet se convirtió en una red internacional, se creó un conjunto de dominios de dos letras correspondientes a los dominios de jerarquía superior en cada país, por ejemplo :

ar **Argentina**
au **Australia**
bz **Belice**
br **Brasil**
ca **Canadá**
cl **Chile**
cn **China**
de **Alemania**
es **España**
gr **Grecia**
it **Italia**
jp **Japón**
mx **México**
ni **Nicaragua**
pa **Panamá**
pl **Polonia**
uk **Reino Unido**
us **Estados Unidos**

Cuando se utiliza un nombre como **balandra.uabcs.mx**, la computadora necesita convertir el nombre de la dirección en una dirección numérica, para lo cual empieza a hacer peticiones de ayuda a los servidores DNS, empezando por el que se encuentra en el extremo derecho del nombre, y recorriendo la dirección hacia la izquierda. Primero se pregunta por la dirección al servidor DNS local, existiendo tres posibilidades :

1.- El servidor local conoce la dirección, debido a que ésta se encuentra en la parte local de la base de datos mundial contenida en el servidor.

2.- El servidor local conoce la dirección solicitada porque alguien más la solicitó recientemente. Cuando se solicita una dirección, el servidor DNS la guarda durante algún tiempo, por si alguien la necesita posteriormente.

3.- El servidor local no conoce la dirección, pero sabe como buscarla . Para esto, el software del servidor local sabe cómo comunicarse con el **servidor raíz**. El servidor raíz conoce todas las direcciones de los servidores de nombres que tienen a su cargo las direcciones de jerarquía superior (por ejemplo **mx**). Un servidor de nombres pregunta al servidor raiz la dirección del servidor responsable de la zona **uabcs**, y con esta información se comunica con ese servidor y le pide la dirección del servidor **balandra**, que es la computadora con la que se quería comunicar.

La gran ventaja del DNS es que permite partir en segmentos manejables la gigantesca red mundial Internet. Además, aunque la red tiene muchas computadoras, todas tienen nombres únicos y éstos se encuentran organizados de tal manera que es posible recordarlos con cierta facilidad [KROLE94].

2.2.- Ventajas del uso de tutoriales en Internet.

Cuando se tiene material informático que debe compartirse entre un grupo grande de integrantes de una organización, conviene manejarlo por medio de una red de computadoras. Si además existe una gran dispersión entre los integrantes de una organización, debe pensarse en una red que cubra todas las localidades donde se encuentre al menos un integrante, aunque esté equipado con una máquina portátil. Si además queremos permitir que inclusive quienes no pertenezcan a la organización puedan tener acceso a la información que generamos, la red deberá cubrir la mayor cantidad de localidades existentes y deberá ser la más accesible.

Los tutoriales propuestos en este documento caen dentro de esa categoría de material informático, y la única red existente en la actualidad que puede garantizar tal cobertura es Internet.

Capítulo 3.- Tutoriales básicos.

Utilizaremos el término *tutoriales básicos* para referirnos a aquellos que se manejan a través de WWW y que se elaboran utilizando única y exclusivamente el lenguaje HTML.

En lo que resta de este capítulo, describiremos algunas herramientas utilizadas para la elaboración y el manejo de tutoriales básicos, a la vez que propondremos algunos criterios de diseño.

3.1.- WWW.

Para manejar los tutoriales en Internet, es necesario utilizar un sistema de distribución tal como WWW (World Wide Web, que en lo sucesivo simplemente llamaremos Web). Este sistema fue desarrollado inicialmente por Tim Barners Lee, un estudiante del Laboratorio Europeo de Física de Partículas (CERN), ubicado en Suiza. Web está basada en hipertexto, el cual consiste en documentos electrónicos cuyo texto contiene partes que actúan como enlaces o vínculos hacia otros documentos con información más detallada.

Aunque el concepto de hipertexto tiene más de 30 años, lo novedoso en su uso por parte de WWW consiste en la manera que lo aplica a la gigantesca base de información de Internet, convirtiéndose en un verdadero sistema hipermedia, en el que las páginas permiten acceder a imágenes, sonidos y videos ; con lo que se ha incrementado enormemente el atractivo de Web.

Además de ser gráfica y fácil de manejar, WWW es interactiva, soporta cualquier plataforma y puede tener acceso a muchos tipos de información (http, ftp, Gopher, etc).

3.1.1.- Visualizadores para Web.

El acceso a Web se logra por medio de un cierto tipo de software denominado **visualizador** (browser).

Por medio del visualizador, la computadora **cliente** del usuario obtiene información almacenada en otras computadoras llamadas **servidores**.

La manera en que trabaja un visualizador es la siguiente :

1.- Dado un **URL** (Uniform Resource Locator, Localizador Uniforme de Recurso) , que es un apuntador hacia una porción de información,
2.- debe ser capaz de tener acceso a la información, u operar de la forma deseada, basándose en los contenidos del URL.

La información apuntada por un URL puede ser de diferentes tipos, dependiendo del protocolo utilizado. Algunos de estos protocolos son :

HTTP (HiperText Transfer Protocol, Protocolo para la Transferencia de HiperTexto) , utilizado para los documentos Web.

FTP (File Transfer Protocol, Protocolo para la Transferencia de Archivo)

FILE Protocolo para referirse a archivos del disco local.

MAILTO Protocolo para el envío de correo electrónico.

3.1.1.1.- Visualizadores existentes.

Aunque Web se empezó a utilizar con visualizadores que solamente manejaban texto (tal como Lynx), actualmente predominan los que ofrecen capacidades para el manejo de gráficos.

Aquí no se pretende realizar una descripción exhaustiva de todos los visualizadores existentes, sino describir brevemente algunos de los más populares tales como :

Mosaic.

Mosaic fué el primer visualizador gráfico a todo color, y posee el mérito de haber popularizado a la Web. Mosaic pertenece a la Universidad de Illinois, pero existen versiones comerciales para X Windows System, Microsoft Windows y Macintosh.

Netscape.

El Netscape Navigator (de Netscape Communications Corporation) desplazó a Mosaic en cuanto a popularidad, llegándose a convertir en el visualizador a vencer por parte de las empresas desarrolladoras de software que ven en Internet una poderosa y amplia vía para la actividad comercial a nivel mundial.

Netscape está disponible para Microsoft Windows, Macintosh y X Windows System, y aunque no es un producto de distribución gratuita, puede conseguirse una copia de evaluación en **ftp ://ftp.netscape.com** .

En la Figura 3.1 se muestra la apariencia del Netscape Navigator Gold 3.01 para Windows 95.

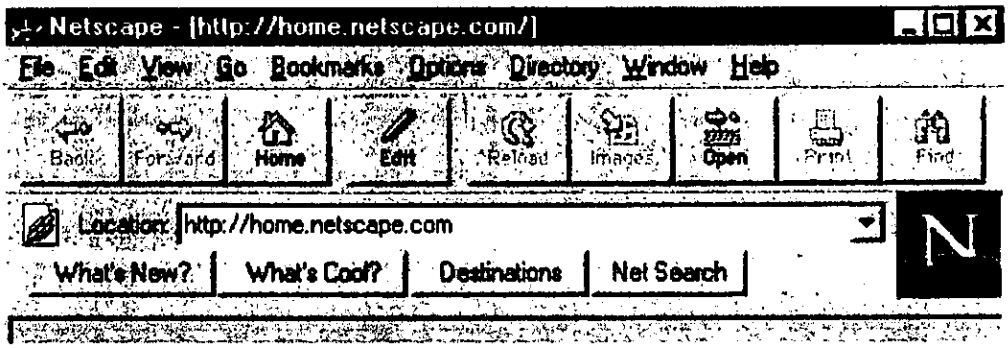


Figura 3.1.- Netscape Navigator Gold 3.01 para Windows 95.

Internet Explorer.

Microsoft lanzó en agosto de 1995 (junto con Windows 95) un visualizador llamado Internet Explorer (Explorador de Internet). Basado originalmente en la tecnología de Mosaic, Internet Explorer se ha convertido rápidamente en un visualizador muy completo que está compitiendo fuertemente contra Netscape por la supremacía del mercado sustentado en Windows 95.

Mientras tanto, cualquiera puede obtener una copia del Internet Explorer a través de Web utilizando el URL **http ://www.microsoft.com/windows/ie/iexplorer.htm**

3.1.2.- Partes de un URL.

Aunque por lo general los visualizadores nos evitan la molestia de tener que recordar las cadenas completas que representan a los URLs, conviene saber el significado de cada una de sus partes, principalmente cuando se quiere comunicar a otras personas la ubicación de cierta información.

Generalmente, los URLs constan de: el protocolo, el nombre del anfitrión (host) y el directorio y el nombre del archivo, como se ejemplifica en la Figura 3.2.

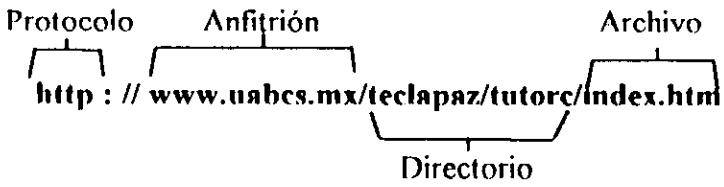


Figura 3.2.- Partes de un URL.

3.1.4.- Presentaciones Web.

Una **presentación Web** es un conjunto de una o más **páginas Web** que contienen texto y/o gráficos que, vinculados de cierta forma, producen el efecto deseado.

A las páginas Web también se les conoce como **documentos Web** y consisten de archivos únicos en disco que se recuperan y se presentan en un formato adecuado mediante visualizadores Web.

La página de entrada a una aplicación Web se conoce como **página base** (home page) y es lo primero que aparece en pantalla cuando se ejecuta una aplicación. El papel asignado a la página base es similar al de la portada de un libro o a la fachada de un local comercial, puesto que determina la primera impresión para quien visita el lugar. Por ejemplo, cuando se carga un visualizador, normalmente aparece la página base de la empresa que lo desarrolló. En el caso de los proveedores de información para Web (como es el caso de los desarrolladores de tutoriales), la página base es el punto de acceso a las páginas que conforman la aplicación . El URL de la página base es lo que se proporciona a las personas cuando se les invita a visitar un sitio Web, por ejemplo :

http ://www.uabcs.mx/tutorc/index.htm

3.1.5.- Organización de las presentaciones Web.

Es muy importante que las páginas de una aplicación correspondan a una estructura bien planeada (de acuerdo al tipo de información que se va a presentar) y que cubra objetivos bien establecidos.

Existen varias formas de organización para las presentaciones, entre otras : la **lineal**, la **jerárquica** y la **de red**.

En nuestro caso, el objetivo es elaborar páginas Web que presenten información correspondiente a tutoriales del tipo CAI descrito en el Capítulo 1. Por la naturaleza de este tipo de tutoriales, la organización más conveniente para ellos es la jerárquica, con una página base que sirve como punto de entrada y con los temas y subtemas dispuestos en forma de menú, como se muestra en la Figura 3.3.

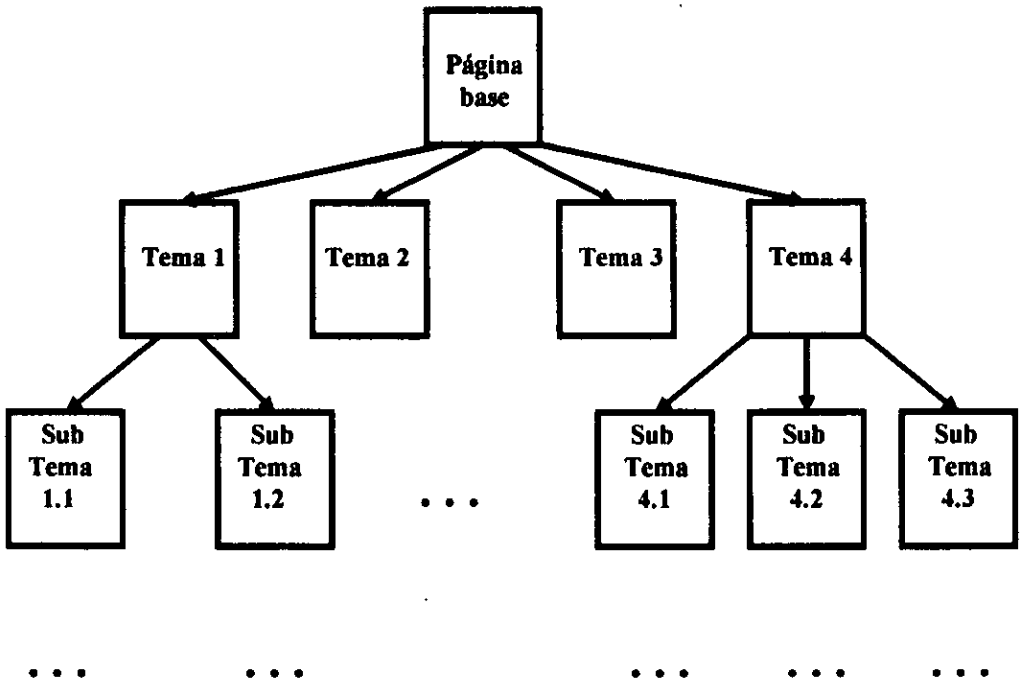


Figura 3.3.- Organización jerárquica.

Un tutorial de tipo CAI corresponde al material contenido en un libro de texto, por lo que la página base es análoga a la portada del libro, las páginas intermedias semejan los índices de cada capítulo, y las páginas inferiores de la jerarquía corresponden al texto o contenido temático. Cada página se almacena individualmente en un archivo.

3.2.- HTML.

Para producir documentos que puedan manejarse en la Web, deberá utilizarse un lenguaje como HTML (HyperText Markup Language, Lenguaje de Marcado de HiperTexto). Los nombres de los archivos de texto que contengan código HTML deberán tener la extensión HTM o HTML, cuidando las convenciones establecidas por el sistema operativo que se esté usando en el servidor donde se instalen dichos archivos. Por ejemplo, en Unix se hace una diferenciación entre las letras mayúsculas y las minúsculas que aparecen en los nombres de los archivos de manera que : **indice.htm** es distinto de **Indice.htm** y de **INDICE .HTM** . En todas las referencias escritas en los guiones HTML deberán usarse los nombre de los archivos **exactamente** como se crearon en el sistema de archivos.

3.2.1.- Código permitido en HTML.

Los archivos generados con la sintaxis de HTML deberán ser archivos que contengan sólo caracteres del código ASCII (7 bits) y no podrá incluirse ninguno de los caracteres del conjunto "extendido" (8 bits), ya que cada plataforma tiene una definición diferente de los caracteres en el rango superior del código ASCII.

Los visualizadores que soportan HTML interpretan los códigos del rango superior del código ASCII como caracteres pertenecientes al conjunto ISO-Latin-1 (un superconjunto del código ASCII).

Se espera que en el futuro el estándar de HTML permita utilizar un conjunto de caracteres más amplio, como por ejemplo Unicode [LEMAL96].

3.2.2.- Caracteres especiales.

El estándar de HTML define un conjunto de códigos denominados **entidades de carácter** , las cuales tienen el siguiente formato :

&cadena ;

Esto es, todas las entidades de carácter empiezan con un símbolo de ampersand (&) y terminan con un punto y coma.

Dependiendo del valor de **cadena**, las entidades de carácter se clasifican en **nombradas** y **numeradas**.

Las cadenas de las entidades de carácter nombradas tienen significados que pueden recordarse con cierta facilidad, por ejemplo si un archivo contiene la secuencia :

&oacute ;

el visualizador desplegará :

ó

ya que **acute** es la cadena que denomina al acento agudo que se utiliza en Español.

Por otra parte, las cadenas de entidades de carácter numeradas utilizan el simbolo de numeral (#), seguido de un número decimal que corresponde al código del carácter en el conjunto ISO-Latin-1. Así, para obtener la letra **o con acento agudo** del ejemplo anterior, puede escribirse :

&#243 ;

Utilizando las entidades de carácter, podemos escribir archivos HTML con texto en Español, por ejemplo :

" Debemos esforzarnos para edificar la patria que añoramos para nuestros ni&#241o;s, libre de violencia y corrupción "

Esto será desplegado por un visualizador en la siguiente forma :

" Debemos esforzarnos para edificar la patria que añoramos para nuestros niños, libre de violencia y corrupción "

Nótese que la **ñ** se ha obtenido con las dos formas existentes. La mejor forma será la que represente el menor esfuerzo para quien elabora el documento.

3.2.3.- Caracteres reservados.

Existe un grupo de caracteres que tienen un significado especial en el lenguaje HTML, por lo que su uso no está permitido como parte del contenido temático de un archivo de hipertexto. Por lo tanto, para que los visualizadores puedan desplegar esos caracteres reservados, deberán utilizarse las entidades de carácter correspondientes.

La tabla 3.1 muestra las entidades y los caracteres reservados en HTML.

Entidad	Carácter
&lt	<
&gt	>
&amp	&
&quot	"

Tabla 3.1.- Caracteres reservados en HTML.

3.2.4.- Estructura de los archivos HTML.

La base de la estructura de un archivo HTML la constituyen las **etiquetas** (tags), y el texto que representa el contenido temático del documento .

3.2.4.1.- Etiquetas.

El visualizador interpreta las instrucciones representadas por las etiquetas y gestiona ante el sistema operativo la realización de las tareas requeridas. Por esto, aunque el lenguaje HTML es independiente de las plataformas, deberá utilizarse un visualizador específico para cada una de ellas ; teniendo así implementaciones de visualizadores para UNIX, Macintosh, Windows, etc. De esta manera, la función del visualizador se puede ver como la de un gestor de recursos entre el usuario y el sistema operativo, además del papel que adopta como formateador en la presentación de los resultados.

Existe un Grupo de Trabajo para el Lenguaje de Marcado de Hipertexto (Hypertext Markup Language Working Group) que se encarga de formular y publicar la especificación para el lenguaje HTML. La versión de especificación vigente a la fecha es la 2.0 (aunque ya se cuenta con el borrador de la versión 3.0, de la cual se han implementado algunas características en algunos visualizadores).

Las etiquetas tienen como característica común el empezar con el carácter < y finalizar con > .

En lo que resta esta sección solo se describirán brevemente algunas de las principales etiquetas para HTML. Para mayores detalles puede consultar [LEMAL96] y [BERNE94].

<HTML> .... </HTML>	Delimitan todo el documento.

La etiqueta <HTML> deberá ser lo primero que se escriba en un documento HTML, para indicar que su contenido está escrito en ese lenguaje.
La etiqueta </HTML> complementa a <HTML> y marca el final del documento.

<HEAD> ... </HEAD>	Delimitan el encabezado del documento.

La etiqueta <HEAD> especifica el inicio del encabezado o prólogo de un documento, mientras que </HEAD> marca el final del encabezado.

<TITLE> ... </TITLE>	Delimitan el título del documento.

Es conveniente escribir un título para indicar desde el principio de lo que trata el documento. El título es utilizado por programas que clasifican las páginas Web y otros.

El complemento de la etiqueta <TITLE> es </TITLE>.

Veamos un ejemplo utilizando las etiquetas anteriores :

```
<HTML>
<HEAD>
<TITLE>Tutorial de C++</TITLE>
</HEAD>
   ...
   ...
</HTML>
```

Las etiquetas pueden escribirse con mayúsculas o minúsculas, y no es necesario colocarlas en una posición específica, por lo que el anterior ejemplo pudo haberse escrito así:

```
<html><head><title>Tutorial de C++</title></head>
  ...
  ...
</html>
```

Como puede observarse, lo único que hay que cuidar es el orden en que se escriben las etiquetas de complemento.

<BODY> ... </BODY>	Delimitan el cuerpo del documento.

El cuerpo de un documento es el lugar donde se escribe todo el contenido temático, así como las etiquetas y vínculos.

La etiqueta <BODY> debe escribirse después de la etiqueta de cierre de encabezado </HEAD>. La etiqueta de cierre </BODY> normalmente se escribe antes de la etiqueta </HTML>.

Ejemplo :

```
<HTML>
<HEAD>
<TITLE>Tutorial de C++</TITLE>
</HEAD>
<BODY>
```

Este tutorial de Lenguaje C++ puede ser &uacutetil para las personas que desconocen completamente el lenguaje, hasta personas que posean un nivel intermedio. El tiempo estimado para dominar un lenguaje de programaci&oacuten es variable, dependiendo de la dedicaci&oacuten que se tenga. En t&eacuterminos generales, medio a&ntildeo ser&aacute suficiente para lograr un dominio a nivel intermedio.

```
</BODY>
</HTML>
```

No se preocupe en acomodar el texto entre <BODY> y </BODY> ; el visualizador lo desplegará en el espacio disponible, justificándolo hacia la izquierda, aún en el caso de que usted lo tecleara en una sola línea (esto es, sin utilizar caracteres de nueva línea de su editor de textos).

La etiqueta <BODY> tiene los siguientes atributos :

BACKGROUND = "imagen" Usa una imagen como mosaico para el fondo de la página.

BGCOLOR = "color" Establece el color para el fondo de la página.

TEXT = "color" Establece el color para el texto.

LINK = "color" Establece el color para los vínculos.

ALINK = "color" Establece el color para el vínculo activo.

VLINK = "color" Establece el color para los vínculos visitados.

En todos los atributos anteriores, **color** es una cadena que representa la concatenación de tres números en hexadecimal, con el siguiente formato :

"#RRGGBB"

donde :

RR representa la cantidad de color rojo (Red)
GG representa la cantidad de color verde (Green)
BB representa la cantidad de color azul (Blue)

Encabezados : <H1>...</H1> ... <H6>...</H6>

Las etiquetas de encabezado sirven para delimitar los títulos que aparecen en el cuerpo del documento, y pueden utilizarse para jerarquizar los tamaños de las letras de los temas, subtemas, etc..

<H1> ... </H1> Delimitan un encabezado de nivel 1.
<H2> ... </H2> Delimitan un encabezado de nivel 2.
<H3> ... </H3> Delimitan un encabezado de nivel 3.
<H4> ... </H4> Delimitan un encabezado de nivel 4.
<H5> ... </H5> Delimitan un encabezado de nivel 5.
<H6> ... </H6> Delimitan un encabezado de nivel 6.

Cada nivel de encabezado corresponde a un tamaño de letra ; así que los encabezados de nivel 1 tienen el tipo de letra más grande, y los encabezados de tipo 6 el tamaño más pequeño.

Párrafos. <P> ... </P>

La etiqueta <P> se utiliza para el manejo de párrafos en los documentos HTML 3.0, con su correspondiente complemento (optativo) </P>.

Para evitar confusiones, es recomendable utilizar a <P> como separador de párrafos, y evitar el uso de </P>.

Comentarios. < ! >

En HTML existe una manera de indicarle al visualizador que no deberá desplegar cierta parte del documento, sino que será considerada como un comentario para quien analiza el texto fuente.

La secuencia < ! marca el inicio de un comentario, mientras que el final es señalado con >

Formateo de texto y caracteres.

Entre las etiquetas para especificar el formateo de texto y caracteres, las más utilizadas son las siguientes :

**
** Obliga a generar una nueva línea (rompe la línea actual).
<HR> Fuerza un retorno de carro (Hard Return)
<PRE>...<PRE> Delimitan texto preformateado (Utiles para listados de programas)
<B> ... </B> Sirven para delimitar texto en **negrita**.
<I> ... </I> Delimitan texto en *itálica* (cursiva).

Encabezados : <H1>...</H1> ... <H6>...</H6>

Las etiquetas de encabezado sirven para delimitar los títulos que aparecen en el cuerpo del documento, y pueden utilizarse para jerarquizar los tamaños de las letras de los temas, subtemas, etc..

<H1> ... </H1> Delimitan un encabezado de nivel 1.
<H2> ... </H2> Delimitan un encabezado de nivel 2.
<H3> ... </H3> Delimitan un encabezado de nivel 3.
<H4> ... </H4> Delimitan un encabezado de nivel 4.
<H5> ... </H5> Delimitan un encabezado de nivel 5.
<H6> ... </H6> Delimitan un encabezado de nivel 6.

Cada nivel de encabezado corresponde a un tamaño de letra ; así que los encabezados de nivel 1 tienen el tipo de letra más grande, y los encabezados de tipo 6 el tamaño más pequeño.

Párrafos. <P> ... </P>

La etiqueta <P> se utiliza para el manejo de párrafos en los documentos HTML 3.0, con su correspondiente complemento (optativo) </P>.

Para evitar confusiones, es recomendable utilizar a <P> como separador de párrafos, y evitar el uso de </P>.

Comentarios. < ! >

En HTML existe una manera de indicarle al visualizador que no deberá desplegar cierta parte del documento, sino que será considerada como un comentario para quien analiza el texto fuente.

La secuencia < ! marca el inicio de un comentario, mientras que el final es señalado con >

Formateo de texto y caracteres.

Entre las etiquetas para especificar el formateo de texto y caracteres, las más utilizadas son las siguientes :

**
** Obliga a generar una nueva línea (rompe la línea actual).
<HR> Fuerza un retorno de carro (Hard Return)
<PRE>...<PRE> Delimitan texto preformateado (Utiles para listados de programas)
<B> ... </B> Sirven para delimitar texto en **negrita**.
<I> ... </I> Delimitan texto en *itálica* (cursiva).

Imágenes. <IMG>

La etiqueta <IMG> sirve para desplegar una imagen y tiene, entre otros, los siguientes atributos :

SRC="nombre_imagen" Especifica el nombre del archivo donde se almacena la imagen a desplegar.

ALT="texto" Especifica la cadena de texto alternativo que se desplegará en los visualizadores que no soportan imágenes.

ALIGN="posición" Determina la alineación de la imagen, donde **posición** puede ser : **LEFT, CENTER , RIGHT**

Vínculos. <A> ... </A>

Los vínculos (enlaces, ligas , links) se manejan por medio de las etiquetas **<A>** y **</A>**. El nombre de estas etiquetas tiene su origen en la palabra del idioma Inglés **Anchor** (Ancla). Entre las etiquetas de apertura y de cierre debe escribirse el texto que será considerado como vínculo.

La etiqueta de apertura <A> tiene el **atributo** :

HREF="nombre_de_archivo" Especifica la referencia a un archivo de hipertexto, donde **nombre_de_archivo** puede incluir la trayectoria.
En caso de no especificarse la trayectoria, se supone que el archivo se encuentra almacenado en el mismo directorio que el documento donde se escribió la etiqueta de vínculo.

Además de texto, pueden utilizarse imágenes como vínculos, como se ve en el siguiente ejemplo :

<A HREF="t11.htm"><IMG SRC="bola.gif"BORDER=0>1.1.- Estructura de un programa en C++.</A>

En este caso, tanto la imagen **bola.gif** como la cadena de texto **1.1.- Estructura de un programa en C++.** servirán para vincular hacia el archivo **t11.htm**

Mapas de imágenes.

Los mapas de imágenes consisten de una imagen donde cada parte de ella funciona como un vínculo hacia diferentes páginas.

Las etiquetas que se utilizan para la creación de mapas de imágenes son **<MAP>** , **</MAP>** , **<AREA>** e **<IMG>.**

Ejemplo :

```
<MAP NAME="linea">
<AREA SHAPE="rect" COORDS="304,0,464,20" HREF="portada.htm">
</MAP>
<IMG SRC="linea.gif" BORDER=0 USEMAP="#linea" >
```

En este caso, se crea un mapa cuyo nombre es **linea**. La forma del área del mapa es un rectángulo cuyos vértices superior izquierdo e inferior derecho tienen las coordenadas (x,y) **(304,0)** y **(464,20)** , respectivamente. La etiqueta <MAP> necesita su complemento </MAP>. Por medio de este mapa se establece un vínculo hacia el archivo **portada.htm** .

Por último, se utiliza la etiqueta <IMG> para desplegar la imagen almacenada en el archivo **linea.gif** y asociarla con el mapa denominado **linea**.

Cuadros.

Una página Web puede dividirse en cuadros, en los cuales se pueden desplegar, de manera independiente y simultánea, diferentes documentos. Si el tamaño del cuadro no es suficiente para desplegar el documento completo, este se podrá visualizar en su totalidad utilizando las barras de desplazamiento. El usuario podrá pasar de un cuadro a otro con solo accionar el botón izquierdo del ratón sobre la superficie del cuadro al que desea cambiarse.

Las etiquetas que se usan para el manejo de cuadros son :

<FRAMESET> y su complemento **</FRAMESET>** y,

<FRAME>.

La etiqueta **<FRAMESET>** sirve para establecer el número de cuadros en que va a dividirse una ventana, así como el tamaño de cada uno de ellos. Esta etiqueta tiene los siguientes atributos :

ROWS = "lista de valores" Divide la página en renglones.

COLS = "lista de valores" Divide la página en columnas.

Donde **lista de valores** indica el tamaño del cuadro en :

- **valor entero**
- **valor de porcentaje**
- **valor relativo**

El **valor entero** establecerá el tamaño del cuadro en **pixeles**.

El **valor de porcentaje** establece los tamaños de los cuadros como un **porcentaje**. La suma de los porcentajes deberán sumar 100 (En caso contrario, el visualizador forzará a que los cuadros llenen la ventana completa).

Por ejemplo :

<FRAMESET COLS="80% , 20%"> divide la página en dos cuadros columna. El cuadro de la izquierda tomará el 80 por ciento de la ventana y el cuadro de la derecha el 20 por ciento.

El **valor relativo** se implica por medio del asterisco (*), y especifica el tamaño de un cuadro en **relación con los otros cuadros**.

Ejemplos :

<FRAMESET ROWS = " *, * ,* " > crea una relación de **3 (1+1+1)**, dividiendo la página en tres renglones de igual tamaño.

<FRAMESET COLS = " 3*, * ,* " > crea una relación de **5 (3+1+1)**, dividiendo la página en tres columnas. A la columna de la extrema izquierda le asigna tres quintas partes del espacio disponible en la ventana, a la columna del centro y a la de la extrema derecha les asigna una quinta parte.

<FRAMESET COLS = " 3*, 2* ,* , * " > crea una relación de **7 (3+2+1+1)**, dividiendo la página en cuatro columnas. A la columna de la extrema izquierda le asigna tres séptimas partes del espacio disponible en la ventana, a la segunda columna de izquierda a derecha le asigna dos séptimas partes y a las dos columnas de la extrema derecha les asigna una séptima parte.

Por lo general, es más fácil utilizar valores de porcentaje o mezclas de éstos con los otros tipos de valores.

Por ejemplo :

<FRAMESET COLS = " 15% , * > asigna el 15 por ciento del espacio disponible a la columna de la izquierda, y el resto a la columna de la derecha.

La etiqueta **<FRAME>** sirve para especificar el contenido que va a desplegarse en un cuadro, así como otras características relacionadas con su comportamiento. Algunos de los atributos de **<FRAME>** son :

SRC = "fuente" que especifica el archivo donde se almacena el contenido a desplegar en el cuadro.

BORDER = valor que especifica el tamaño del borde para el cuadro. Un valor igual a cero especifica un cuadro sin borde.

NORESIZE que indica que el cuadro no puede ser redimensionado por el usuario.

SCROLLING = opción que especifica si el cuadro tendrá una barra de desplazamiento o no. Los valores para **opción** son :

YES sí habrá barra de desplazamiento.

NO no habrá barra de desplazamiento.

AUTO el visualizador colocará (en caso necesario) una barra de desplazamiento. Este es el valor por omisión.

Formularios.

Los formularios o formas se manejan por medio de la etiqueta **<FORM>**, la cual tiene como complemento a **</FORM>.**

La etiqueta **<FORM>** tiene los siguientes atributos :

ACTION = "URL" , especifica el URL del guión que procesará el formulario actual.

METHOD = "modo_envío" , define la manera en que la información del formulario será enviada al servidor. Los valores para **modo_envío** pueden ser : **POST y GET**.

ENCTYPE = "modo" , establece el modo de encriptamiento para la información que se envía.

3.3.- Gráficos.

Los archivos de gráficos a utilizar en Internet deben ser de formato **GIF** o de formato **JPEG**.

El formato de archivo **GIF** es un formato de 8 bits y es propiedad de **CompuServe** ; pero el algoritmo de compresión que utilizan pertenece a **Unisys**.

El formato de archivo **JPEG** fué desarrollado por el Joint Photographic Experts Group (de cuyas iniciales toma su nombre) y , como no está patentado, su uso no requiere de consideraciones legales. Puesto que sirve para el manejo de gráficos con calidad fotográfica, el formato **JPEG** es de 24 bits .

Debido a su formato de 8 bits, las imágenes **GIF** pueden manejar como máximo 256 colores, mientras que la imágenes **JPEG** pueden manejar un poco más de 16 millones.

Dentro de las ventajas que ofrece el uso de archivos con formato **GIF** pueden mencionarse las siguientes :

- Están soportados por todos los visualizadores de Web .
- Permiten la inclusión de varias imágenes dentro de un solo archivo.
- Pueden proporcionar animación, al manejar dentro de un ciclo las imágenes del archivo.

Las imágenes o gráficos se pueden utilizar tanto para diseñar el fondo de las páginas como para generar vínculos, como se muestra a continuación.

```
1 : <HTML>
2 : <HEAD>
3 : <TITLE> Portada</TITLE>
4 : </HEAD>
5 : <BODY BACKGROUND="bk103.jpg" text="#394A64">
6 : <CENTER><IMG SRC="tuto1.gif"></CENTER>
7 : <CENTER><IMG SRC="logo1.gif"></CENTER><br>
8 : <CENTER><A HREF="acercade.htm" onMouseOver='status="Acerca de" ;
9 :  return(true)' onMouseOut='status="Document: Done"'>
10 :  <IMG border=0  SRC="acercade.gif">
11 : </A></CENTER>
12 : </BODY>
13 : </HTML>
```

Como puede observarse en la línea 5 del código anterior , uno de los atributos de la etiqueta **<BODY>** es **BACKGROUND** el cual tiene un valor igual a **bk103.jpg**. Esto significa que el fondo de la página deberá rellenarse con el gráfico contenido en el archivo **bk103.jpg**. La extensión **.jpg** indica que el archivo tiene un formato **JPEG**.

En la línea 6, la etiqueta **<IMG>** tiene un atributo **SRC="tuto1.gif"**, con lo que deberá desplegarse la imagen contenida en el archivo **tuto1.gif**. La línea 7 produce un efecto similar.

La etiqueta **<A>** de la línea 8 significa que se va a manejar un vínculo. El atributo **HREF** especifica que se va a establecer un vínculo hacia el archivo **acercade.htm**. En la línea 10 se especifica que el vínculo va a estar formado por la imagen contenida en el archivo **acercade.gif**.

3.4.- Gifs animados.

Un gif animado es un archivo de gráficos (grabado en formato **GIF**) que contiene varias imágenes. Para crear un archivo gif se requiere de herramientas especiales, como por ejemplo el **Gif Construction Set** de Alchemy Mindworks Inc. , del cual se puede obtener una copia en **http ://www.mindworkshop.com** .

Por ejemplo, si el archivo **animado.gif** es un gif animado, puede activarse incluyendo en un guión HTML la etiqueta : **<IMG SRC = "animado.gif ">**

Con esto, cada una de las imágenes contenidas en el archivo **animado.gif** serán desplegadas en secuencia y se producirá un efecto de animación durante un determinado número de ciclos.

Para invocar al Gif Construction Set debe pulsarse dos veces el icono  y a continuación aparecerá una ventana como la que se muestra en la Figura 3.4.

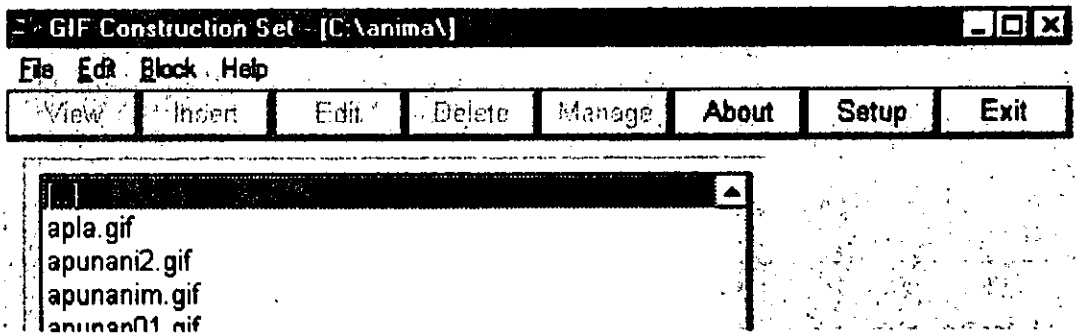


Figura 3.4.- Ventana principal del Gif Construction Set.

En este caso, al hacer doble click sobre el nombre **apunani2.gif** se despliega una ventana como se observa en la Figura 3.5.

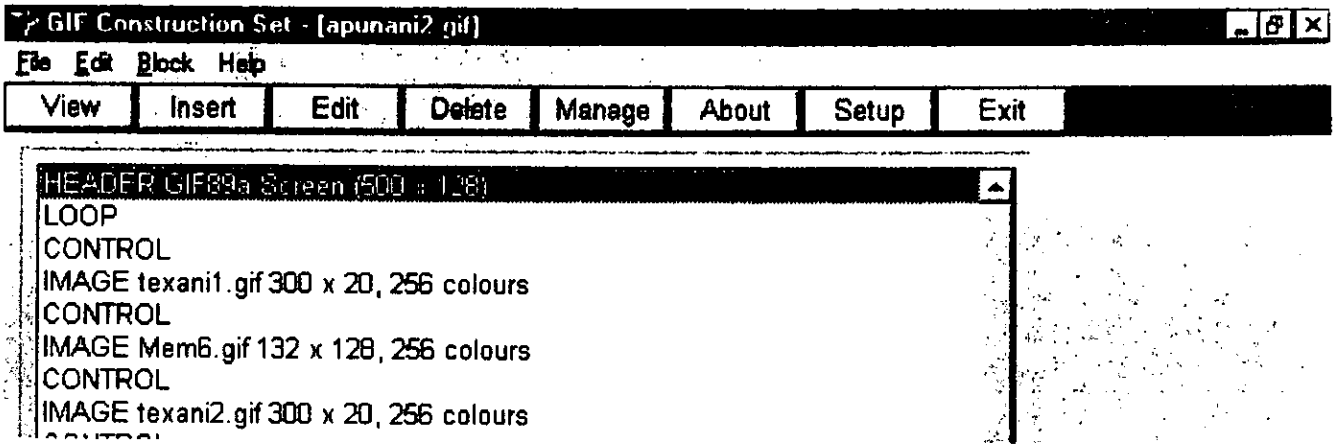


Figura 3.5.- Estructura de control del archivo apunani2.gif

En la Figura 3.5 se observa parte de la estructura de control del archivo **apunani2.gif**, donde la primera línea corresponde al encabezado del archivo y describe el tamaño (en pixeles) del cuadro donde se desplegarán las imágenes.

Haciendo doble click sobre la palabra LOOP de la segunda línea, se abre una ventana que permite establecer el número de veces que se desplegarán las imágenes que forman la animación.

De la línea 3 en adelante se observan parejas de líneas CONTROL / IMAGE, donde cada línea de control corresponde a la imagen especificada en la línea que le sigue.

Con un doble click en una línea CONTROL, se tiene acceso a una ventana donde se pueden editar algunas características de la imagen correspondiente, como se muestra en la Figura 3.6.

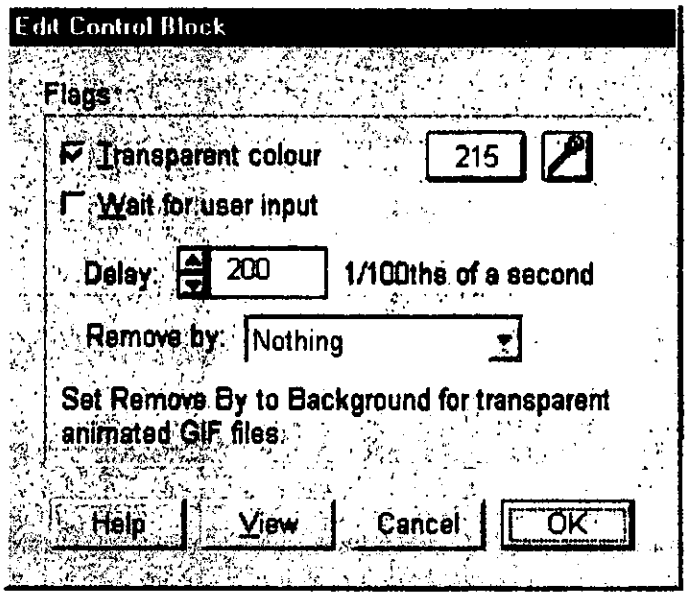


Figura 3.6.- Edición del bloque de control.

Al editar el bloque de control, se puede:

- establecer un color específico como transparente
- indicar que se espere una acción del usuario para desplegar la imagen.
- especificar el tiempo de espera(en centésimas de segundo), para desplegar la siguiente imagen.
- especificar si la imagen va a ser removida por otra, por el fondo, o por nada.

Capítulo 4.- Tutoriales y JavaScript.

En este capítulo se revisarán algunas herramientas que permitan mejorar la interacción entre el ser humano y la computadora.

Los tutoriales manejados por medio de la computadora son interactivos por naturaleza, ya que en ellos la participación del aprendiz es indispensable. Sin embargo, en los tutoriales implementados por medio del lenguaje HTML existen ciertos niveles de interactividad que no es posible lograr, haciéndose necesaria la inserción de guiones escritos en JavaScript o la utilización de pequeñas aplicaciones escritas en Java.

La interacción consiste en un diálogo bilateral entre el usuario y la computadora. y se produce cuando el usuario utiliza el teclado, el ratón o algún otro dispositivo de señalamiento para indicar una acción o para proporcionar un dato a la máquina.

4.1.- Programación Orientada a Objetos.

JavaScript es un lenguaje de programación basado en objetos y Java es un lenguaje de programación orientado a objetos (Java se trata en el Capítulo 5), por lo que es necesario revisar los conceptos básicos de la Programación Orientada a Objetos antes de adentrarse en el estudio de ambos lenguajes.

4.1.1.- Objetos.

Un **objeto** es una abstracción que trata de emular a los objetos del mundo real, para lo cual deben considerarse tres aspectos acerca de ellos: su **estado**, su **comportamiento** y su **identidad** [DECKR93].

Tanto en **JavaScript** como en **Java** :

- el **estado** de un objeto está representado por los **datos o atributos**,
- su **comportamiento** por los **métodos** y
- su **identidad** por un **nombre** o **identificador** asignado.

4.1.2.- Clases.

Los lenguajes orientados a objetos permiten describir un prototipo para un conjunto de objetos al que se le conoce como una **clase**.

4.1.2.1.- Instanciación.

La instanciación consiste de la creación de nuevos objetos a partir de una clase dada, por lo que los términos instancia y objeto pueden tomarse como sinónimos.

Para crear un objeto de cierta clase, ésta debe incluir tanto la declaración como la definición (implementación) de un método especial denominado **constructor**.

4.1.3.- Herencia.

Es frecuente que se consideren varias clases de manera que se establezca una relación de herencia (similar a la herencia genética que se da entre las especies animales) donde las subclases heredan tanto el estado como el comportamiento de las superclases, pudiéndose establecer **jerarquías de clases**.

Dentro de una jerarquía de clases puede considerarse una relación de dependencia entre las **subclases** y las **superclases**, tal como se da entre padres e hijos en un árbol genealógico. En la Figura 4.1. se muestra la estructura de una jerarquía de clases cualquiera.

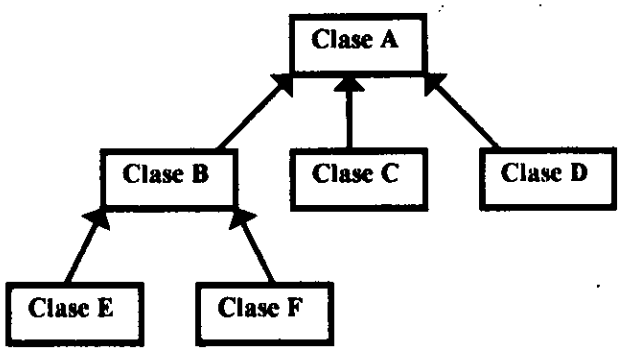


Figura 4.1.- Jerarquía de clases.

La clase A es la superclase de las clases B, C y D ; mientras que la clase B es la sueperclase de las clases E y F.

Por otra parte, las clases B, C y D son subclases de la clase A.

Las **flechas indican relaciones** del tipo : **"hereda de"**. Considerando que las subclases se derivan de las superclases, tales relaciones pueden expresarse como :**"deriva de"**. En el caso del lenguaje Java, donde se considera que las subclases extienden la jerarquía, las relaciones serían : **"extiende a"** .

En una subclase no es necesario definir los atributos y comportamientos definidos en la superclase ; debido al mecanismo de herencia, las subclases tendrán las características de todas las clases antecesoras.

Cuando se crea una instancia de una clase, se obtiene espacio tanto para las variables definidas en las clases antecesoras como para las variables definidas en la clase actual.

4.1.4.- Superposición de métodos.

Los métodos declarados en una superclase podrán definirse en ella o en la subclase. Cuando un método definido en la superclase se vuelve a definir en la subclase, se dice que el método se ha **superpuesto**. Por medio de la superposición, un método puede adaptarse para cubrir los requerimientos de la subclase.

La superposición funciona de la siguiente manera : cuando se invoca un método de un objeto , primero se revisa en la clase del objeto para buscar la definición de ese método ; si no se encuentra, se revisa en la superclase inmediata superior ; de no encontrarse ahí, se busca sucesivamente en las clases superiores de la jerarquía, hasta encontrar la definición buscada . Esto es, para cada método siempre se utiliza la definición que se encuentra primero al recorrer la jerarquía de abajo hacia arriba.

4.1.5.- Notación de punto.

Para indicar que una variable o un método pertenecen a un objeto específico, se utiliza la **notación de punto** que conste en escribir los nombres del objeto y de las variables y los métodos separados por un punto. Por ejemplo, si existe un objeto denominado **carro** , se puede invocar a su método **enciende()** con la siguiente instrucción:

carro.enciende() ;

4.2.- JavaScript.

JavaScript es un lenguaje para el manejo de código inserto en los guiones HTML y se originó a partir de un lenguaje para guiones de Netscape denominado LiveScript.

4.2.1.- Estructura del lenguaje.

Para definir la estructura de un lenguaje de programación , es necesario describir la forma en que maneja los diferentes tipos de datos, los operadores ; además de la sintaxis para las expresiones y para cada una de las estructuras de control.

4.2.1.1.- Tipos de datos.

JavaScript es un lenguaje de programación de **tipos libres**, debido a que las variables pueden tomar y cambiar su tipo de acuerdo al contexto, sin necesidad de asociarles un identificador de tipo.

Los datos en JavaScript pueden ser : números, cadenas, booleanos y el valor nulo.

4.2.1.1.1.- Números

Los números pueden ser **enteros** o **de punto flotante**, según se escriban sin punto decimal, con punto decimal explícito o en formato exponencial. Además, los números pueden ser positivos o negativos.

Enteros.

Los números enteros se escriben sin punto decimal y pueden representarse en cualquiera de los siguientes formatos :

Decimal (Base 10) escrito sin un cero al inicio. Por ejemplo : 213

Octal (Base 8) escrito con un cero al inicio. Por ejemplo : 042

Hexadecimal (Base 16) escrito con **0x** o **0X** al inicio. Por ejemplo : 0x3A

De punto Flotante.

Los números de punto flotante pueden escribirse :

con punto decimal explícito : 3.1416
en formato exponencial : 31416E-4 (31416 por 10 elevado a la menos 4)

4.2.1.1.2.- Cadenas.

Una cadena consta de cero o más caracteres delimitados por comillas dobles o comilla sencillas.

Por ejemplo, las siguientes son dos formas equivalentes de escribir la misma cadena .

- "Esta es una cadena"
- 'Esta es una cadena'

El uso alternativo de los dos tipos de comillas hace posible incluir alguno de los dos dentro de una cadena, como por ejemplo en :

' Visite el taller "Don Cuco" '

4.2.1.1.3.- Booleanos.

Los únicos datos booleanos permitidos en JavaScript son :

false (falso) y

true (verdadero)

4.2.1.1.4.- Valor nulo.

Para representar el valor nulo (el cual significa nada), se utiliza la palabra **null** .

4.2.1.2.- Variables.

Las variables son espacios de memoria donde se almacenan valores que pueden ser reemplazados durante la ejecución de un programa.

4.2.1.2.1.- Declaración de variables.

Una declaración de variable consiste en asociar un nombre con el espacio de memoria reservado para ella. Ese nombre podrá utilizarse en el resto del guión para utilizar el valor actual de la variable o para modificarlo.

El primer carácter de los nombres o identificadores para las variables debe ser una letra o el carácter de subrayado. Los caracteres subsecuentes pueden ser letras, números o el carácter de subrayado.

Se distingue entre letras mayúsculas y minúsculas, por lo que los identificadores :

Valor
VALOR
valor

se considerarán asociados a distintas variables.

Una variable en JavaScript puede declararse de las siguientes maneras :

a).- Utilizando la palabra **var** .

```
Ejemplos :   var  calif ;          // Sin asignar valor inicial.
             var calif = 100 ;  // Asignando valor inicial.
```

b).- Al utilizar el nombre por primera vez.

```
Ejemplo :  calif = 100 ;
```

Aunque la declaración de una variable en JavaScript no es obligatoria, es muy recomendable a fin de lograr un buen estilo de programación.

En JavaScript las variables no requieren de una definición explícita de tipo, pues lo toman en forma dinámica cuando son utilizadas por primera vez [LEMAP96].

4.2.1.3.- Operadores.

Los operadores sirven para combinar uno o mas operandos y producir un resultado. Dependiendo de los tipos de los operandos que utilizan y del tipo de resultado que producen, los operadores pueden incluirse en los siguientes grupos :

4.2.1.3.1.- Operadores aritméticos.

Los operadores aritméticos se aplican sobre operadores numéricos y pueden clasificarse en dos grupos : binarios (que se aplican a dos operandos) y unitarios (aplicados sobre un solo operando).

Los operadores aritméticos binarios sirven para realizar las operaciones básicas de suma, resta, multiplicación y división ; además de la operación módulo, que calcula el residuo de una división entre valores enteros. Los símbolos que identifican a los operadores aritméticos binarios son :

+ para la suma,
- para la resta,
* para la multiplicación,
/ para la división y
% para la operación módulo.

Los símbolos utilizados para representar los operadores aritméticos unitarios son :

- operador de negación , que cambia el signo del valor del operando.
++ operador de incremento, que incrementa en 1 el valor del operando.
-- operador de decremento, que decrementa en 1 el valor del operando.

4.2.1.3.2.- Operadores relacionales.

Los operadores relacionales generalmente sirven para comparar los valores que resultan de evaluar dos expresiones ; aunque también pueden utilizarse para comparar valores de cadenas de caracteres. El resultado es un valor booleano que se representa por las palabras :

true (verdadero) y
false (falso).

En la Tabla 4.1 se describen los operadores relacionales.

Operador	Descripción
== **Igual que**	regresa **true** si los valores de los dos operandos son iguales
!= **Diferente que**	Regresa **true** si los valores de los dos operandos son diferentes.
> **Mayor que**	Regresa **true** si el valor del operando izquierdo es mayor que el valor del operando derecho.
< **Menor que**	Regresa **true** si el valor del operando izquierdo es menor que el valor del operando derecho.
>= **Mayor o igual que**	Regresa **true** si el valor del operando izquierdo es mayor o igual que el valor del operando derecho..
<= **Menor o igual que**	Regresa **true** si el valor del operando izquierdo es menor o igual que el valor del operando derecho.

Tabla 4.1.- Operadores relacionales.

Ejemplos :

100 == 100 regresa **true**
30 < 20 regresa **false**
45 >= 25 regresa **true**
"ella" != "el" regresa **true**

4.2.1.3.3.- Operadores lógicos.

Los operadores lógicos se utilizan sobre operandos de tipo booleano, dando como resultado valores de ese mismo tipo.

El la tabla 4.2 se describen los operadores lógicos.

Operador	Descripción
&& Conjunción (y)	Regresa **true** si el valor de los dos operandos es **true**.
|| Disyunción (o)	Solo regresa **false** cuando el valor de ambos operandos es **false**.
! Negación (no)	Regresa **true** si el valor del operando es **false**. Regresa **false** si el valor del operando es **true**.

Tabla 4.2.- Operadores lógicos.

Como podrá observarse, los operadores de conjunción y de disyunción son binarios. El operador de negación es unitario.

4.2.1.3.4.- Operadores de asignación.

Los operadores de asignación sirven para almacenar un valor en una variable, y se forman utilizando el símbolo = , según se muestra en la Tabla 4.3.

Operador	Descripción
=	Asigna el valor del operando derecho al operando izquierdo.
+=	Suma los valores de los operandos izquierdo y derecho, y el resultado lo asigna al operando izquierdo.
-=	Resta el valor del operando derecho al valor del operando izquierdo y asigna el resultado al operando izquierdo.
*=	Multiplica los valores de los operandos izquierdo y derecho, y asigna el resultado al operando izquierdo.
/=	Divide el valor del operando izquierdo entre el valor del operando derecho y asigna el resultado al operando izquierdo.
%=	Divide el valor del operando izquierdo entre el valor del operando derecho y asigna el residuo al operando izquierdo.

Tabla 4.3.- Operadores de asignación.

4.2.1.3.5.- Operador condicional.

El operador condicional simula la estructura de control **if - else** (descrita más adelante), y se forma con los símbolos **? :**

La sintaxis para expresiones que utilizan este operador es :

(condición) ? valor1 : valor2

Si al evaluar la **condición** se obtiene un resultado igual a **true**, el operador condicional da como resultado el **valor1**. En otro caso, da como resultado **valor2** .

Ejemplo :

((dia_semana >= "Lunes") && (dia_semana<="Viernes")) ? "A trabajar" : "A descansar"

4.2.1.3.6.- Operadores a nivel de bits.

Los operadores a nivel de bits actúan sobre valores enteros de 32 bits en su representación binaria (la que utiliza únicamente ceros y unos) y se describen brevemente en la Tabla 4.4.

Operador	Descripción
~	Produce el complemento del operando (a nivel de bits).
<<	Desplaza los bits hacia la izquierda.
>>	Desplaza los bits hacia la derecha.
>>>	Desplaza los bits hacia la derecha, sustituyendo con ceros los espacios que se desocupan en el extremo izquierdo.
&	Conjunción lógica a nivel de bits (and).
^	Disyunción exclusiva a nivel de bits(xor)
|	Disyunción a nivel de bits (or)

Tabla 4.4.- Operadores a nivel de bits.

4.2.1.4.- Estructuras de control.

Las estructuras de control sirven para controlar el orden de ejecución de las instrucciones que forman un programa y, de acuerdo a su funcionamiento, pueden agruparse en estructuras de :

- Secuencia
- Selección
- Iteración

4.2.1.4.1.- Estructuras de secuencia.

Las estructuras de secuencia consisten de instrucciones que se ejecutan una a continuación de otra, sin bifurcaciones ni iteraciones.

4.2.1.4.2.- Estructuras de selección.

Las estructuras de selección contienen bifurcaciones que se controlan por medio de la evaluación de una condición.

JavaScript cuenta con una estructura de selección if-else , la cual se rige por la sintaxis mostrada a continuación :

if (condición) bloque 1
else bloque 2

donde :

condición es una expresión booleana,
bloque 1 y **bloque 2** son bloques de una o más instrucciones y
la parte **else** es opcional.

Ejemplo :

```
if( x < 10 )
  {
    y = x + 5 ;
    x++ ;
  }
else
   {
     y =  x -5 ;
     x-- ;
   }
```

4.2.1.4.3.- Estructuras de iteración.

Las estructuras de iteración sirven para ejecutar repetidamente un bloque de instrucciones.

JavaScript incluye las siguientes estructuras de iteración :

4.2.1.4.3.1.- La estructura for

Crea una iteración por medio de tres partes encerradas entre paréntesis y separadas una de otra por medio de un punto y coma.

La sintaxis para esta estructura es :

for(inicia ; condición ; actualiza) bloque

donde :

inicia es la parte donde se asignan los valores iniciales a los elementos que sirven como controladores de ciclos,
condición es una expresión booleana ,
actualiza es la parte donde se incrementan o decrementan los controladores, y
bloque es un conjunto formado por una o más instrucciones a ejecutar en cada ciclo.

Ejemplo :

```
for(var i=1 ; i <= 100 ; i++)
  {
    utilidad+=10 ;
    precio =  costo + utilidad;
    capacidad_compra = salario/precio ;
  }
```

4.2.1.4.3.2.- La estructura for-in

Esta estructura de control permite iterar sobre todas las propiedades de un objeto en particular. Se rige por la sintaxis :

for(var propiedad in objeto) bloque

donde :

propiedad es una variable, y **objeto** es una instancia de una clase.
bloque es el conjunto de instrucciones a ejecutar en cada ciclo.

Ejemplo :

Si tenemos una clase denominada **persona**, podemos escribir el siguiente código que despliega los nombres y los valores de todas las propiedades del objeto **miPersona** :

```
var miPersona = new persona() ;
for(var prop in miPersona)
  document.writeln("miPersona." + prop + " = " + miPersona[prop] ) ;
```

Aquí cabe hacer mención sobre la existencia de dos maneras de accesar propiedades de objetos : la notación de punto y el uso de índices asociativos.

Por ejemplo, si el objeto **miPersona** tiene la propiedad **nombre**, entonces se puede hacer referencia a esa propiedad utilizando cualquiera de las siguientes notaciones :

1.- **miPersona.nombre**
2.- **miPersona["nombre"]**
3.- var x = "nombre" ;
miPersona[x]

4.2.1.4.3.3.- La estructura while

Ejecuta un conjunto de instrucciones **mientras** se cumpla una **condición** dada. Cuando la condición produce un valor false, la ejecución continúa en la instrucción que sigue al bloque de la estructura while. La sintaxis para esta estructura es :

while(condición) bloque

Ejemplo :
```
nuevo = 10 ;
while( nuevo > 0 )
  { nuevo-- ;
    acum += nuevo ;
  }
```

4.2.2.- Objetos en JavaScript.

A diferencia de otros lenguajes basados en objetos, JavaScript no permite la extensión de clases (subclasificación) , sino que brinda un conjunto reducido de clases. Además, proporciona un conjunto de objetos integrados que brindan información acerca de la página Web cargada en ese momento y acerca de la sesión actual del visualizador .

4.2.2.1.- Objetos del visualizador.

La mayoría de los objetos integrados en JavaScript forma parte de la jerarquía de objetos del visualizador. Esta jerarquía está construida con base al objeto **window**, como se muestra en la Figura 4.2.

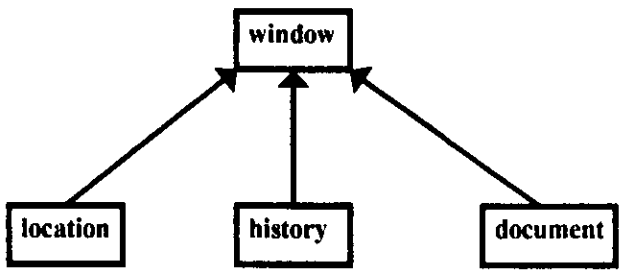


Figura 4.2.- Jerarquía de objetos del visualizador.

A continuación se describen brevemente los objetos de la Figura 4.2.

4.2.2.1.1.- El objeto window.

El objeto **window** de JavaScript es el que tiene la precedencia más alta en relación con la ventana del visualizador que se encuentre abierta. El visualizador crea el objeto window al momento de cargar una página que contiene propiedades que se aplican a toda la ventana .

Dado que su existencia se presupone, no se tiene que escribir el nombre de la ventana al hacer referencia a sus objetos, propiedades o métodos. Por ejemplo, para desplegar un mensaje en la línea de estado se puede hacer de cualquiera de las dos formas siguientes:

```
window.status = "Acerca de"

status = "Acerca de"
```

Las propiedades del objeto **window** se describen en la Tabla 4.1.

Propiedad	Descripción
defaultStatus	Valor de cadena que contiene el valor predeterminado que se despliega en la barra de estado.
frames	Arreglo de objetos para cada cuadro en una ventana. Los cuadros aparecen en el arreglo en el mismo orden que en el código HTML.
length	Valor entero que indica el número de cuadros en una ventana madre.
name	Valor de cadena que contiene el nombre de la ventana o cuadro.
parent	Nombre alternativo para la ventana que contiene la etiqueta <FRAMESET> con que se creó la ventana actual.
self	Nombre alternativo para la ventana actual.
top	Nombre alternativo para la ventana superior.
window	Nombre alternativo para la ventana actual.

Tabla 4.1.- Propiedades del objeto window.

Los métodos del objeto **window** se describen en la Tabla 4.2.

Método	Descripción
alert(mensaje)	Despliega el **mensaje** en el cuadro de diálogo del visualizador.
Close()	Cierra la ventana.
confirm(mensaje)	Despliega el **mensaje** en el cuadro de diálogo del visualizador donde se incluyen los botones **OK** y **CANCEL.** Cuando se elige el botón OK, devuelve **true** y devuelve **false** en caso de elegir CANCEL.
Open(URL, **nombre,** **características)**	Abre el **URL** en la ventana **nombre**. Si **nombre** no existe, se crea una nueva ventana. **características** es una cadena opcional que contiene los siguientes pares nombre=valor : **toolbar=valor** Agregar barra de herramientas ?. **location=valor** Agregar campo de ubicación ?. **directories=valor** Agregar botones de directorio ?. **status=valor** Debe tener barra de estado ?. **menubar=valor** Debe tener barra de menú ?. **scrollbars=valor** Debe tener barra de desplaza-miento ?. **resizable=valor** Puede redimensionarse ?. **width=pixeles** Anchura de la ventana en pixeles. **heigth=pixeles** Altura de la ventana en pixeles. donde **valor** puede ser : **yes, no, 1, 0**
prompt (mensaje, respuesta)	Despliega el **mensaje** en el cuadro de diálogo con un campo de entrada de texto, con el valor predeterminado **respuesta**.
setTimeout(expresión, tiempo)	Evalúa la **expresión** después de un **tiempo** dado en milisegundos. Puede asignarse un nombre para la invocación a este método. Por ejemplo : **nombre**=setTimeout(a=b+c, 5000)
clearTimeout(nombre)	Cancela el tiempo fuera identificado con **nombre**.

Tabla 4.2.- Métodos del objeto window.

Además, el objeto **window** cuenta con los siguientes **manejadores de eventos** (Los eventos son señales generadas cuando ocurren acciones específicas, como pulsar una tecla o un botón del ratón. Un manejador de evento es una variable que almacena valores de un evento dado) :

- **OnLoad** especifica el código JavaScript a ejecutar, cuando la ventana o cuadro terminen de cargarse.
- **OnUnload** especifica el código JavaScript a ejecutar, cuando se abandona la ventana o cuadro.

4.2.2.1.2.- El objeto location.

El objeto **location** contiene información acerca del URL del documento actual. Este objeto se utiliza para determinar el URL de cualquier documento activo, incluyendo aquellos que se encuentran en otras ventanas o cuadros.

Las propiedades del objeto **location** se describen en la Tabla 4.3.

Propiedad	Descripción
hash	Valor de cadena que contiene el nombre del ancla en el URL.
Host	Valor de cadena que contiene el nombre del anfitrión y el número del puerto del URL.
Hostname	Valor de cadena que contiene el nombre de dominio (o dirección numérica IP) del URL.
Href	Valor de cadena que contiene el URL completo.
Pathname	Valor de cadena que especifica la parte del URL correspondiente a la ruta.
port	Valor de cadena que contiene el número de puerto del URL.
protocol	Valor de cadena que contiene el protocolo del URL (hasta antes de las dos diagonales)
search	Valor de cadena que contiene la información pasada a una llamada GET CGI-BIN (información después del signo de interrogación).

Tabla 4.3.- Propiedades del objeto location.

4.2.2.1.3.- El objeto history.

El objeto **history** contiene información acerca de los URLs de las páginas visitadas con anterioridad. Sus métodos se utilizan para navegar hacia cualquier punto en la lista.

La única propiedad de este objeto es **length**, que representa el número de elementos de la lista del historial.

Los métodos del objeto **history** son :

back() que regresa al documento anterior en la lista del historial,

forward() que va hacia adelante, al siguiente documento en la lista del historial, y

go(ubicación) que va al documento especificado por **ubicación** en la lista del historial . El valor de **ubicación** puede ser una cadena o un valor entero positivo o negativo.

4.2.2.1.4.- El objeto document.

Este es uno de los objetos base de JavaScript ya que contiene información acerca de anclas, vínculos, formularios, título, ubicación actual, URL y colores del documento actual.

El objeto **document** es creado por el visualizador al momento de cargar una página (precisamente al evaluar la etiqueta <BODY>) y existe mientras la página esté cargada.

Las propiedades del objeto **document** se describen en la Tabla 4.4.

Propiedad	Descripción
alinkColor	Color de vínculos activos representado como una cadena o una tripleta en hexadecimal.
anchors	Arreglo de objetos de ancla dispuestos en el orden en que aparecen en el documento HTML. Para obtener el número de anclas en el documento, se escribe : anchors.length .
bgColor	Color del fondo del documento.
cookie	Valor de cadena que contiene valores de **galleta** para el documento actual. Una **galleta** es una especie de marca por medio de la cual el servidor almacena información del lado del cliente. Cuando el usuario solicite de nuevo la misma página, el visualizador enviará la galleta al servidor para que encuentre más rápido la página solicitada.
fgColor	Color del primer plano del documento.
forms	Arreglo de objetos de formularios dispuestos en el orden en que aparecen en el archivo HTML. Para obtener el número de formularios en un documento escríbase : **forms.length** .
lastModified	Valor de cadena que contiene la fecha de la última modificación del documento.
linkColor	Color de los vínculos escrito como una cadena o como una tripleta en hexadecimal.
links	Arreglo de objetos de vínculo dispuestos en el orden en que aparecen en el documento HTML. Para obtener el número de vínculos en un documento, escriba : **links.length** .
location	Cadena que contiene el URL del documento actual.
referrer	Valor de cadena que contiene el URL del documento invocador, cuando el usuario sigue un vínculo.
title	Cadena que contiene el título del documento actual.
vlinkColor	Color de los vínculos ya visitados, expresado como una cadena o una tripleta en hexadecimal.

Tabla 4.4.- Propiedades del objeto document.

Los métodos del objeto **document** son :

clear() que despeja la ventana del documento,

close() que cierra el flujo de salida actual,

open(mimeType) que abre un flujo para permitir a los métodos write() y writeln() escribir en la ventana del documento. El argumento mimeType es una cadena opcional que especifica un tipo de documento soportado por el visualizador ,

write() que despliega texto guiones HTML en el documento especificado, y

writeln() que despliega texto guiones HTML en el documento especificado, seguido de un carácter de nueva línea.

Además de los objetos de la jerarquía del navegador, JavaScript proporciona los siguientes objetos que no están relacionados con las ventanas actuales o los documentos cargados.

string : brinda propiedades y métodos para el manejo de literales y variables de cadenas de caracteres.

Math : permite el cálculo de funciones trigonométricas, logaritmos, raices, etc.

Date : proporciona mecanismos para trabajar con fechas y horas.

4.2.2.2.- Instanciación en JavaScript.

Aunque JavaScript no permite la creación de nuevas clases, proporciona un mecanismo de plantilla para la creación de métodos constructores para objetos específicos.

Este mecanismo de plantilla se maneja a través de la palabra clave **function** ,como se muestra en el siguiente ejemplo :

Supongamos que necesitamos almacenar las calificaciones de un grupo de alumnos . Lo primero que necesitamos es implementar el constructor para los objetos **alumno** :

```
function alumno( nombre, calif )
{
  this.nombre = nombre ;
  this.calif = calif ;
```

En este caso, a los atributos del objeto actual **this** (este) se le asignan los valores de los argumentos **nombre** y **calif** que se le pasaron al método constructor **alumno**.

Utilizando la palabra reservada **new** , ahora se pueden crear instancias de la clase **alumno.**

PerezGJ = **new** alumno("Pérez García, Juan" , 95) ;

Para referirnos a los atributos del objeto referido por PerezGJ, podemos utilizar cualquiera de las siguientes notaciones :

PerezGJ.nombre ,cuyo valor es : "Pérez García, Juan" , se puede escribir como **PerezGJ["nombre"]** , o bien como **PerezGJ[0]**

Un objeto puede contener otros objetos, por lo que es posible incluir objetos como parte de la definición de un método constructor.

Por ejemplo, podemos manejar información acerca del proyecto en que está trabajando un alumno. Para esto, primero definimos un constructor denominado **proyecto :**

```
function proyecto ( titulo, fechaEntrega, alumno )
{
  this.titulo = titulo ;
  this.fechaE = fechaEntrega ;
  this.alumno = alumno;
}
```

Con el objeto **PerezGJ**, ahora podemos crear un nuevo objeto referenciado por **proyTut** :

proyTut = new proyecto("Elaboración de Tutoriales", "15/DIC/97", PerezGJ) ;

Con esto, ahora se puede hacer referencia a los atributos de **PerezGJ** dentro de **proyTut** , así :

```
proyTut.titulo
proyTut.fechaE
proyTut.PerezGJ.nombre
proyTut.PerezGJ.calif
```

Un objeto de la clase **function** también puede regresar un valor (de manera similar a las funciones de otros lenguajes de programación) utilizando la palabra clave **return**.

4.2.3.- Guiones JavaScript.

Un **guión JavaScript** es un trozo de código escrito en lenguaje JavaScript.

JavaScript es un lenguaje interpretado, por lo que el visualizador requiere que el código fuente del guión JavaScript se inserte en un documento HTML.

Para que el visualizador reconozca el texto correspondiente a un guión escrito en JavaScript, debe utilizarse la etiqueta **<SCRIPT>**. Opcionalmente, se puede especificar el lenguaje que se va a utilizar, como se observa en la siguiente plantilla de ejemplo :

```
<SCRIPT LANGUAGE="JavaScript">
  ...
  Programa en JavaScript
  ...
</SCRIPT>
```

Se obtiene el mismo resultado al escribir la etiqueta <SCRIPT> en cualquiera de las siguientes formas :

```
<SCRIPT LANGUAGE="JavaScript">
<SCRIPT LANGUAGE="LiveScript">
<SCRIPT>
```

Ya que el código JavaScript debe insertarse en un guión HTML, por el momento no existe la posibilidad de ocultarlo de cualquier persona que visite las páginas donde aparezca dicho código. Lo que sí es posible es ocultarlo de los visualizadores que no soportan JavaScript (no por razones de mantener el código en secreto, sino para evitar que dichos visualizadores desplieguen texto confuso).

Para ocultar el código a los visualizadores que no soportan JavaScript deben hacerse los siguientes cambios a la plantilla mostrada en el ejemplo anterior :

```
<SCRIPT LANGUAGE="JavaScript">

< !-- Oculta el guión a otros visualizadores
  ...
  Programa en JavaScript
  ...

// Termina de ocultar el guión a otros visualizadores  -->

</SCRIPT>
```

Como puede observarse en la plantilla modificada, el ocultamiento empieza con la secuencia de caracteres:

<!

y finaliza con:

>

Los visualizadores que no soportan JavaScript interpretarán como un comentario todo lo que encuentren entre <! y >.

Capítulo 5.- Tutoriales mejorados con Java.

Los tutoriales elaborados con las herramientas descritas hasta el capítulo anterior tienen limitaciones sobre todo en lo referente a la interacción humano-máquina. Además, el usuario dispone en todo momento del código fuente de los guiones escritos en HTML y en JavaScript. Si el usuario de los tutoriales necesita una mayor interacción con la máquina y el programador decide mantener la privacidad de su código fuente, el lenguaje de programación Java es una buena alternativa. En este capítulo se revisarán las características más relevantes del lenguaje Java, enfatizando en aquellas que nos ayuden a elaborar tutoriales que permitan una mayor interacción con el usuario.

5.1.- Características del lenguaje Java.

Java es un lenguaje de programación orientado a objetos que, entre otras, tiene las características de ser :

Simple

Comparado con C++, Java es más simple[VANHA96] , puesto que no tiene :

- Sobrecarga de operadores
- Archivos de cabecera
- Preprocesador
- Aritmética de apuntadores
- Estructuras
- Uniones
- Arreglos multidimensionales
- Plantillas
- Conversión implícita de tipos

Orientado a Objetos

Con la excepción de los tipos simples tales como números y booleanos, la mayoría de las cosas en Java son objetos. El código de Java está organizado en clases y solo maneja herencia sencilla. En la raíz de la jerarquía de clases siempre está la clase **Object** [VANHA96].

Distribuido

Java está diseñado para soportar aplicaciones en redes, por lo que se dice que es un lenguaje distribuido. Soporta varios niveles de conectividad por medio de las clases contenidas en el paquete **java.net**.

Compilado e interpretado

Aunque puede parecer contradictorio, Java puede clasificarse como un lenguaje compilado y también como uno interpretado. Esto se debe a que, en una primera fase, el código fuente se compila para producir **código de bytes** (bytecode , un código muy cercano al código de máquina), el cual puede transportarse sin ningún problema a computadoras de diversas arquitecturas. Una vez que se encuentra en la máquina cliente, el código de bytes es traducido y ejecutado por un programa intérprete denominado **Máquina Virtual de Java** .

Robusto

Java elimina ciertos tipos de errores de programación, por lo que facilita la escritura de programas confiables. También es un lenguaje muy estricto con los tipos, lo que permite revisiones extensas en tiempo de compilación para encontrar problemas potenciales de mezcla de tipos. Por otra parte, al no soportar apuntadores elimina la posibilidad de sobre escribir la memoria y corromper los datos ; la asignación dinámica de memoria la realiza por medio de los métodos constructores de objetos, y los bloques de memoria desocupados son recuperados por su **recolector automático de basura**.

El **manejo de excepciones** es otra característica de Java que permite la creación de programas más robustos. Una excepción es una señal para alguna clase de condición excepcional, tal como un error [FLAND96].

Seguro

Los visualizadores utilizan la verificación del código de bytes para asegurar que los programas no contienen virus.

Independiente de la arquitectura

Puesto que Java está diseñado para crear aplicaciones basadas en Internet, el código de bytes de éstas debe poder ejecutarse en cualquiera de las computadoras que conforman la red. No importa la clase de máquina en que se compiló el código y no importa la clase de máquina donde va a ejecutarse. La única condición es que la máquina en que va a ejecutarse el código cuente con una **Máquina Virtual de Java** [FLAND96].

Multi-hilos

Java provee soporte para múltiples hilos de ejecución que pueden manejar diferentes tareas ; de manera tal que una aplicación puede estar reproduciendo un archivo de sonido y desplegando una página, mientras que en el fondo se carga un archivo de imagen [FLAND96].

Dinámicamente extensible

Es posible construir interfaces de Java para bibliotecas que fueron escritas en otros lenguajes. Esto es muy fácil puesto que las estructuras de datos escritas en Java son muy similares a los tipos de datos y estructuras de C. El gran problema es que la mayoría de las bibliotecas existentes no son multi-hilos.

Un programa Java puede declarar ciertos métodos para que sean **native** (nativo). Después, estos métodos nativos se mapean a funciones definidas en bibliotecas de programas que se enlazan dinámicamente a la máquina virtual. Por razones de seguridad, usualmente esta característica se desactiva para programas que se cargan a través de la red [VANHA96].

5.2.- Estructura del lenguaje Java.

En esta sección se describirá la forma en que Java maneja los tipos de datos, los operadores, la sintaxis para las expresiones, y la sintaxis para cada una de las estructuras de control.

5.2.1.- Tipos de datos.

Java cuenta con ocho tipos de datos primitivos para números enteros, números de punto flotante, caracteres, y valores booleanos.

5.2.1.1.- Números enteros.

En Java se manejan cuatro tipos de datos enteros. Todos ellos pueden ser positivos o negativos, como se observa en la Tabla 5.1.

Tipo	Tamaño en bits	Rango de valores permitidos
byte	**8**	**-128 a 127**
short	**16**	**-32,768 a 32,767**
int	**32**	**-2,147,483,648 a 2,147483,647**
long	**64**	**-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807**

Tabla 5.1.- Tipos de datos enteros.

5.2.1.2.- Números de punto flotante.

Los números de punto flotante se manejan con un punto decimal y cumplen con el estándar IEEE754. En Java existen dos tipos de números de punto flotante :

1. **float** de **32 bits**, precisión sencilla, y
2. **double** de **64 bits**, precisión doble.

5.2.1.3.- Caracteres.

Para caracteres individuales, Java utiliza el tipo **char**, con **16 bits** de precisión (sin signo) ; suficientes para manejar el conjunto de caracteres **Unicode**.

Unicode es un estándar que permite dar soporte a caracteres de diferentes idiomas. Este nuevo estándar se creó para cubrir las deficiencias del código ASCII en el manejo de caracteres pertenecientes a idiomas distintos del Inglés.

5.2.1.4.- Booleanos.

Los únicos valores permitidos para el tipo **boolean** son :

true (verdadero) y
false (falso).

Además de los ocho tipos primitivos (simples) descritos anteriormente, en Java se pueden manejar nuevos tipos (simples o compuestos) por medio de la creación de nuevas clases (tal como las clases predefinidas **String**, **Font** y **OvalShape**).

5.2.1.5.- Tipos de datos de referencia.

Los tipos de datos no primitivos en Java son los objetos y los arreglos. Frecuentemente, estos tipos no primitivos son denominados "tipos de referencia" porque se manejan por referencia y no por valor como se hace con los tipos primitivos. En el manejo por referencia, las variables almacenan direcciones de memoria y no valores literales.

En los casos de las asignaciones y del paso de argumentos en los métodos, los tipos primitivos son pasados por valor ; los objetos y los arreglos son pasados por referencia.

Ejemplo 1 :

```
int i = 10 ;  // La variable i almacena el valor 10 .

int j = i ;  // Se asigna a la variable j el valor almacenado actualmente en i  ( 10 ).

i = 20 ;  // Ahora i almacena el valor 20 , y j almacena el valor 10 .
```

Ejemplo 2 :

```
Button x,y ;          // Crea las referencias x , y a objetos de la clase Button.

x = new Button() ;  // Crea un nuevo objeto con el constructor de la clase Button ;
                    // la dirección del objeto se asigna a la variable x.

y = x ;               // A la variable y se le asigna la misma dirección que tiene
                      // almacenada la variable x ( La variable y se referirá al mismo
                      // objeto que x ) .

x.setLabel("Correcto !") ; // Se modifica el objeto a través de x.

String z = y.getLabel( ) ;  // La variable z tendrá el valor :  Correcto !.
```

En Java no es necesario destruir los objetos que ya no se utilizan, puesto que el lenguaje usa una técnica denominada **recolección de basura** que detecta automáticamente los objetos que ya no están utilizándose. Un objeto ya no se utiliza cuando dejan de existir referencias a él.

5.2.1.6.- Arreglos.

Un arreglo está formado por un conjunto de elementos del mismo tipo que utilizan el mismo nombre pero que se distinguen entre ellos por medio de un índice.

A diferencia de otros lenguajes, Java maneja los arreglos como objetos. Por esta razón, los siguientes conceptos aplicados a los tipos de referencia y a los objetos también los aplica a los arreglos.

- Son manipulados por referencia.
- Se crean dinámicamente por medio de la palabra clave **new**.
- Son destruidos automáticamente por el recolector de basura.

5.2.1.6.1.- Creación de arreglos.

Existen dos formas para crear arreglos en Java.

La primera utiliza la palabra clave **new**, y especifica el número de elementos que van a existir en el arreglo.

Ejemplo :

```
int arregloDeCincoEnteros[ ] = new int[5] ;
```

Los elementos de los arreglos creados de esta manera se inicializan al valor predeterminado para su tipo. Por ejemplo, los elementos de tipo **int** se inicializan a **0** y los **objetos** se inicializan a **null**.

La segunda forma de crear un arreglo es inicializando estáticamente los valores de los elementos.

Ejemplo :

```
int arregloDeCincoEnteros[ ] = { 10, 15, 20, 25, 30 } ;
```

En este caso se crea dinámicamente un arreglo y se inicializan sus elementos a los valores especificados.

5.2.1.6.2.- Acceso a los elementos de un arreglo.

Para acceder a los elementos de un arreglo, deben especificarse el nombre del arreglo y el índice que identifica al elemento.

Los índices son números enteros que representan la posición de cada elemento en el arreglo. Al índice del primer elemento en el arreglo le corresponde el valor cero.

Así, del ejemplo anterior tenemos que :

arregloDeCincoEnteros[0] tiene un valor igual a **10**,
arregloDeCincoEnteros[1] tiene un valor igual a **15**,
arregloDeCincoEnteros[2] tiene un valor igual a **20**,
arregloDeCincoEnteros[3] tiene un valor igual a **25** y
arregloDeCincoEnteros[4] tiene un valor igual a **30**.

Observe que el valor del índice del último elemento es igual al tamaño del arreglo menos uno (esto se debe a que la numeración empieza en cero).

5.2.1.6.3.- Arreglos multidimensionales.

Si definimos un arreglo multidimensional como aquel que permite el uso de varios índices (donde cada índice representa una dimensión) , entonces podemos afirmar que Java soporta arreglos multidimensionales y que los implementa con arreglos de arreglos, de manera similar a como lo hacen otros lenguajes.

Ejemplo :

```
byte arregloBidimensional[][] = new byte [20][30] ;
```

Esto crea un **arreglo de arreglos de bytes**, al reservar dinámicamente (en tiempo de ejecución) **600 bytes** de memoria (en el montículo - heap - de la memoria RAM). La dirección de ese bloque de memoria se asigna a la variable **arregloBidimensional**.

No es necesario especificar los valores de cada una de los índices de un arreglo, por lo que podemos escribir el ejemplo anterior como :

```
byte arregloBidimensional[][] = new byte [600][] ;
```

Lo importante es que se especifique el total de elementos y señalar las dimensiones necesarias.

La regla para este caso es que las primeras **n** dimensiones (n > = 1) deben tener el número de elementos especificados, y que esas dimensiones deben estar seguidas por **m** dimensiones adicionales, las cuales no tendrán valores especificos.

De acuerdo a esta regla, es correcto escribir :

String quinceCadenas[][][][][] = new String[5][3][][][] ;

pero no :

String quinceCadenas[][][][][] = new String[5][][][][3] ;

5.2.2.- Comentarios.

Java maneja tres tipos de comentarios :

1.- El que puede abarcar más de una línea, y que empieza con /* y termina con */ , como se muestra en el siguiente ejemplo :

```
/* Este es un comentario que
   abarca varias líneas y sirve
   para documentar de manera extensa
   un programa */
```

2.- El que abarca parte de o toda una línea. Empieza con dos diagonales // y finaliza con un carácter de retorno de carro (enter).

Ejemplos :

```
// Este es un comentario de una línea completa.

x = 10 ; // Comentario que abarca parte de una línea.
```

3.- El comentario que empieza con /** y finaliza con */ es considerado por el compilador exactamente igual que el primer tipo. La diferencia consiste en que puede ser utilizado por **javadoc** para documentar aplicaciones.

5.2.3.- Literales.

Las literales son valores que se toman tal cual, que utilizan su propio espacio de memoria y que no están asociados a un identificador. Generalmente se usan para asignarlas a variables de tipos específicos o para utilizarlas como parámetros al invocar ciertos métodos.

Podemos clasificar a las literales en los siguientes grupos :

5.2.3.1.- Literales numéricas.

Las literales numéricas consisten de números enteros o de punto flotante, por lo que las podemos agrupar como :

5.2.3.1.1.- Literales enteras.

Consisten de valores sin punto decimal, los cuales pueden presentarse en los siguientes formatos :

Decimal (base 10).

Las literales se presentan como grupos de dígitos del **0** al **9**, que **no empiecen con 0**.
Ejemplos : 33, 100 , 1997

Octal (base 8).

Las literales se presentan como grupos de dígitos del **0** al **7**, **empezando con cero**.
Por ejemplo, los números en decimal : 33, 100 y 1997
en octal se representan como : 041, 0144 y 03715 , respectivamente.

Hexadecimal (base 16).

Las literales se presentan como grupos formados por los dígitos del **0** al **9** y por las letras de la **A** a la **F** (o sus correspondientes minúsculas), y **deben empezar con 0x o con 0X**.
Por ejemplo, los números en decimal : 33, 100 y 1997
en hexadecimal se representan como : 0x21, 0x64 y 7CD, respectivamente.

Las literales enteras ocupan, de manera predeterminada, un espacio de 32 bits (que corresponde al tipo int). Puede indicarse un espacio de 64 bits agregando una letra L (o su correspondiente minúscula).

Por ejemplo :
La literal 100 se almacena en 32 bits, pero la literal 100L se almacena en 64 bits.

5.2.3.1.2.- Literales de punto flotante.

Pueden escribirse en dos formatos : con punto decimal explícito o en notación exponencial.

Por ejemplo, la literal con punto decimal explícito : 3.1416
puede representarse, en formato exponencial, como : 31416E-4,
lo cual significa " 31416 multiplicado por :10 elevado a la menos 4" ,
esto es : 31416×10^{-4} (recordando que $10^{-4} = 0.0001$)

Las literales de punto flotante ocupan un espacio de 64 bits (correspondiente al tipo double), pero pueden forzarse a ocupar un espacio de 32 bits (tipo float) agregándoles la letra F (o su correspondiente minúscula).

Por ejemplo : -25.6 se almacena en 64 bits, pero -25.6F se almacena en 32 bits.

5.2.3.2.- Literales booleanas.

Pueden consistir solamente de las palabras clave **true** o **false**.

5.2.3.3.- Literales de caracteres.

Equivalen a un solo carácter y se representan entre comillas sencillas, como por ejemplo 'a', '$', '\n'. Se almacenan como caracteres Unicode (en espacios de 16 bits). Pueden representar explícitamente un carácter o su correspondiente valor en código en formato octal, hexadecima o Unicode. Los caracteres no imprimibles o los valores de los caracteres en Unicode se representan por medio de secuencias de escape, como se muestra en la tabla 5.2.

Secuencia	Significado
\n	**Nueva línea**
\t	**Tabulador horizontal**
\b	**Retroceso**
\r	**Retorno de carro**
\f	**Alimentación de forma**
\\	**Diagonal inversa**
\'	**Comilla sencilla**
\"	**Comillas dobles**
\ddd	**Valor en Octal**
\xdd	**Valor en Hexadecimal**
\udddd	**Unicode**

Tabla 5.2.- Secuencias de escape.

Las letras **d** en las secuencias de la Tabla 5.2 representan dígitos, de acuerdo al formato indicado.

5.2.3.4.- Literales de cadena.

Aunque una cadena es una secuencia de caracteres, en Java no se manejan como simples arreglos de caracteres sino como **instancias de la clase String**. Puesto que las cadenas son objetos, puede determinarse su longitud y tener acceso a cualquiera de los caracteres individuales que las forman.

Las literales de cadena consisten de series de caracteres (inclusive secuencias de escape) encerradas entre comillas dobles.

Ejemplos :

"Esta es una literal de cadena"

" " // <= Ahí está una cadena vacía

"Literal de cadena con \n una nueva línea incluida "

5.2.3.5.- Literal nula.

Existe una literal para indicar la inexistencia de valor y se representa por medio de la palabra reservada **null**. Esta literal no debe confundirse con el valor cero que pertenece a los tipos numéricos ; tampoco debe confundirse con la cadena vacía " ".

5.2.4.- Expresiones.

Las expresiones son mezclas de **operandos** y **operadores**. Cuando se evalúa una expresión, se obtiene un resultado.

Un operando es un dato que puede estar almacenado en una variable, en una constante, o en una literal.

5.2.5.- Operadores.

Los operadores en Java son los mismos que se utilizan en JavaScript y se describieron en la sección 4.2.1.3.

5.2.6.- Variables.

Una variable es un espacio de memoria que tiene asociados un **nombre** y un **tipo** de dato. Además, almacena un **valor** que puede cambiar durante el tiempo de ejecución.

5.2.6.1.- Declaración de variables.

Toda variable deberá declararse antes de utilizarla en cualquier instrucción. En la declaración, se establece el nombre de la variable, su tipo y, de manera opcional, su valor inicial.

La sintaxis básica para la declaración de variables es :

tipo nombre ;

Donde :

tipo es cualquiera de los descritos anteriormente,

nombre es un identificador de variable.

Ejemplo :

float impuesto ;

De manera opcional, se puede declarar más de una variable en una instrucción, separando los nombres por medio de comas.

Ejemplo :

float impuesto, sueldo, servicios ;

De igual manera, es posible asignar valores iniciales a una o a todas las variables en la declaración .

Ejemplo :

float impuesto = 10 , sueldo = 20, servicios ;

El tipo de una variable puede ser :

- alguno de los ocho tipos primitivos,
- el nombre de una clase o interfaz, o
- un arreglo.

Más adelante se tratará lo referente a interfaces y arreglos.

Los nombres (llamados también **identificadores**), deben sujetarse a las siguientes restricciones :

- Sólo pueden incluir : **letras, dígitos numéricos**, el carácter de **subrayado** (_) y el **signo de pesos ($)**
- El carácter **inicial** de un nombre solo puede ser : una **letra**, el carácter de **subrayado** o el **signo de pesos**.

Existe un conjunto de **palabras reservadas** (mostrado en la tabla 5.7) que no pueden utilizarse como nombres de variables, ya que tienen un uso predeterminado.

abstract	**boolean**	**break**	**byte**	**byvalue**	**case**	**cast**
catch	**char**	**class**	**const**	**continue**	**default**	**do**
double	**else**	**extends**	**false**	**final**	**finally**	**float**
for	**future**	**generic**	**goto**	**if**	**implements**	**import**
inner	**instanceof**	**int**	**interface**	**long**	**native**	**new**
null	**operator**	**outer**	**package**	**private**	**protected**	**public**
rest	**return**	**short**	**static**	**super**	**switch**	**synchronized**
this	**throw**	**throws**	**transient**	**true**	**try**	**var**
void	**volatile**	**while**				

Tabla 5.7.- Palabras reservadas en Java.

Existe un grupo de nombres de métodos de la clase **Object** que, aunque no pertenecen a la categoría de palabras reservadas, conviene no utilizarlos como nombres de variables. La Tabla 5.8 presenta esos nombres.

clone	**getClass**	**notifyAll**
equals	**hashCode**	**toString**
finalize	**notify**	**wait**

Tabla 5.8.- Nombres de métodos definidos en la clase Object.

5.2.7.- Constantes.

Una constante representa un espacio de memoria que tiene asociados : un **nombre** o identificador, y un **valor** que **no** puede modificarse en tiempo de ejecución.

Aunque Java reserva la palabra **const**, hasta la fecha no la utiliza para la definición de constantes. Para definir constantes, deben utilizarse las palabras **static** y **final** precediendo al nombre de la constante.

Las palabras **static** y **final** se conocen como modificadores y se describen más adelante en las tablas 5.9 y 5.10 . El orden en que se escriban no es relevante, pudiéndose escribir :

final static tipo nombreConstante = literal ;
o
static final tipo nombreConstante = literal ;

Ejemplo :

static final int IVA = 10 ;

Aunque no es una restricción, se acostumbra escribir los nombres de las constantes solo con letras mayúsculas. Esto permite distinguir fácilmente entre los nombres de las variables y los de las constantes.

5.2.8.- Estructuras de control.

Las estructuras de control sirven para controlar el orden de ejecución de las instrucciones que forman un programa . De acuerdo a su funcionamiento, pueden agruparse en estructuras de **secuencia, selección e iteración**.

5.2.8.1.- Estructuras de secuencia.

Las estructuras de secuencia consisten de instrucciones que se ejecutan una a continuación de otra, sin bifurcaciones ni iteraciones.

5.2.8.2.- Estructuras de selección.

Las estructuras de selección contienen bifurcaciones que se controlan por medio de la evaluación de una condición. Java cuenta con dos estructuras de selección : **if-else** y **switch**.

5.2.8.2.1.- La estructura if-else.

La estructura de selección if-else se rige por la sintaxis mostrada a continuación :

if (condición) bloque 1
else bloque 2

donde :

condición es una expresión booleana,
bloque 1 y **bloque 2** son bloques de una o más instrucciones.

La parte **else** es opcional.

Ejemplo :

```
if( calificacion >= 70 )
  {
    estatus = "Acreditada";
  }
else
    {
     estatus = "No acreditada" ;
    }
```

5.2.8.2.2.-La estructura switch.

La estructura switch es útil cuando se tiene que elegir entre más de dos opciones. En lugar de evaluar una condición que solo produce valores booleanos, esta estructura evalúa una expresión que produce valores enteros.

La sintaxis para la estructura switch es :

```
switch(expresión_entera)
   {
     case(valor1) : bloque1 ; break ;
     case(valor2) : bloque2 ; break ;
         ...          ...        ...
     case(valorN) : bloqueN ; break ;
     default : bloqueX ;
    }
```

En caso de que el resultado de evaluar **expresión_entera** coincida con **valor1**, se ejecuta **bloque1** y se interrumpe (por medio de la orden **break ;**) la ejecución de la estructura switch ; en caso de coincidir con el **valor2**, se ejecuta el **bloque2** y se interrumpe ; y así sucesivamente. La etiqueta **default :** es opcional, pero en caso de existir, y si el valor de **expresión_entera** no coincidió con ninguno de los valores de las opciones, se ejecuta **bloqueX**.

La instrucción **break** sirve para romper la ejecución de la estructura y pasar al siguiente nivel exterior de anidamiento. En caso de omitirse, se ejecutan todos los bloques de instrucciones que siguen a la opción coincidente, hasta encontrar una instrucción **break** o el fin de la estructura **switch**.

El tipo de **valor1**, **valor2**, ... y **valorN** puede ser **byte**, **char**, **short**, **int** o **long**.

Ejemplo :

```
char opera ;
...
switch(opera)
  {
    case '+' : suma(operando1, operando2) ; break ;
    case '-' : resta(operando1, operando2) ; break ;
    case '*' : multiplica(operando1, operando2) ; break ;
    case '/' : divide(operando1, operando2) ; break ;
    default : despliegaError("Operación no permitida...") ;
  }
```

5.2.8.3.- Estructuras de Iteración.

Las estructuras de iteración sirven para ejecutar repetidamente un bloque de instrucciones.

El lenguaje Java ofrece las siguientes estructuras de iteración :

5.2.8.3.1.- La estructura while.

En esta estructura se ejecuta un bloque de instrucciones **cero o más** veces, dependiendo del resultado de evaluar una expresión booleana. Se rige por la siguiente sintaxis :

```
while(condición)
  bloque ;
```

Primero se evalúa la expresión booleana **condición**. En caso de que el resultado de dicha evaluación sea **true**, se ejecuta el **bloque** y vuelve a evaluarse **condición**. Cuando el resultado de evaluar **condición** sea **false**, se interrumpe la ejecución de **bloque** y se continúa con la ejecución de las instrucciones que siguen a la estructura.

Debe tenerse la precaución de incluir en **bloque** una instrucción para cambiar el valor de al menos una de las variables que intervienen en **condición**, de manera tal que el resultado llegue a ser **false** y no se caiga en un **ciclo infinito**.

Ejemplo :

```
double valor = 1.00, precio = 1000.00 ;
while(valor *precio <  precio*3.0 )
  {
    valor =  precio * 1.3 ;
     precio *= 1.1 ;
  }
```

5.2.8.3.2.- La estructura do-while.

La estructura **do-while** ejecuta un bloque de instrucciones y después evalúa una condición, lo que garantiza que el **bloque** de instrucciones se ejecutará al menos una vez, aunque el resultado de evaluar **condición** sea **false**. Se rige por la sintaxis :

```
do {
     bloque ;
   }while(condición) ;
```

Ejemplo :

```
double valor, precio = 1000.00 ;
do{
     valor =  precio * 1.3 ;
      precio *= 1.1 ;
   } while(valor*precio < precio*3.0 ) ;
```

Observe que en, este ejemplo, no es necesario inicializar la variable **valor** .

5.2.8.3.3.- La estructura for.

Esta es la estructura de iteración más versatil, puesto que permite simular una estructura while y también que se declaren variables en la sección de **inicio**.

La sintaxis para esta estructura es :

for(inicio ; condición ; actualiza) bloque

donde :

inicio es la parte donde se asignan los valores iniciales a los elementos que sirven como controladores de ciclos,
condición es una expresión booleana ,
actualiza es la parte donde se incrementan o decrementan los controladores, y
bloque es un conjunto formado por una o más instrucciones a ejecutar en cada ciclo.

La tarea original para la que fue creada esta estructura es para ejecutar un bloque durante un número predeterminado de veces como en :

```
for( int i=0 ; i<10 ; i++)
   {
      x = 2*i ;
      y = x - i ;
    }
```

donde la tarea se ejecuta 10 veces (con valores para i desde 0 hasta 9). Sin embargo, puede comprobarse que el siguiente código arroja los mismos resultados que el ejemplo de la estructura **while** mostrado anteriormente. Se han cambiado los nombres de las variables por sus iniciales para obligar a que el código aparezca en una sola línea. A propósito : ¿Cuántas veces se ejecuta el código ?

```
for( double v = 1.00, p = 1000.00 ; v *p <  p*3.0 ; v =  p * 1.3 , p *= 1.1 ) ;
```

5.2.8.3.4.- Ruptura de estructuras de iteración.

Las estructuras de iteración finalizan cuando, al evaluar la expresión que representa la condición, se obtiene un resultado igual a **false** . Sin embargo, es posible romper la ejecución de estas estructuras antes de que finalicen normalmente. Esto se logra utilizando la palabra clave **break**, de la misma manera que en el caso de la estructura **switch**.

Ejemplo :

```
int valorDado, valorBuscado , int valorMinimo = 100 ;
do{
     valorBuscado = buscaValor( ) ;
     valorDado = daValor( ) ;
     if(valorBuscado == valorDado ) break ;
   }while(valorBuscado >= valorMinimo) ;
```

Cuando se tienen estructuras de iteración anidadas, siempre que se finalice o se rompa la ejecución de una de las más internas, se continúa la ejecución de la estructura envolvente más próxima. Para modificar esto, y obligar a que el salto se dé hacia el exterior de la envolvente más próxima, se utiliza **break** con una **etiqueta**. Una etiqueta es un identificador al cual se le agrega un símbolo de dos puntos (:), como se muestra en el siguiente ejemplo :

```
salto:
for(int i = 1 ; i <= 10 ; i++)
  for(int j = 1 ; j <= 5 ; j++)
    for(int k = 1 ; k <= 3 ; k++)
       {
            System.out.println( "i =  " + i + "  , j = " + j +  " , k = " + k ) ;
            if( (i + j + k) >= 7 )
               break salto ;
       }
 System.out.println("Fin de las iteraciones....") ;
```

En este caso, cuando la suma de **i+j+k** de como resultado 7 se suspenderá la ejecución del las estructuras y se dará un salto fuera de ellas, hasta el nivel de la etiqueta **salto:** , continuando la ejecución en la última línea para desplegar : **Fin de las iteraciones....**

De manera similar a **break**, la instrucción **continue** obliga a que una estructura inicie la siguiente iteración desde el principio (omitiendo en esa iteración las instrucciones que siguen a la línea que contiene la palabra **continue**). También se pueden utilizar etiquetas con la instrucción **continue**.

5.3.- Objetos en Java.

Excepto los tipos primitivos, todo en Java se maneja como objeto. Esto compromete a quien lo utilice a tener una idea precisa acerca del modelo de objetos y acerca de la nomenclatura particular empleada por el lenguaje para adoptar dicho modelo.

A continuación describiremos algunos términos nuevos correspondientes al lenguaje de programación Java, tomando como referencia los conceptos básicos estudiados en la sección 4.1.

5.3.1.- Definición de Clases.

La definición de clases consiste en crear nuevas clases en las cuales se declaran las variables y métodos de la clase. Además, toda clase deberá pertenecer a una jerarquía basada en la herencia sencilla.

La jerarquía de clases en Java tiene, predeterminadamente, como raíz la clase **Object** que se encuentra incluida en el paquete **java.lang** . Todas las clases y jerarquías de clases, incluyendo las creadas por el programador, dependerán (directa o indirectamente) de la clase **Object**.

La notación de punto se utiliza para indicar la pertenencia de una clase a un **paquete** de clases dado. Por ejemplo :

java.lang.Object

indica que la clase **Object** pertenece a un paquete denominado **lang**, el cual a su vez pertenece al paquete **java** .

La definición de una clase involucra la palabra clave **class** , de acuerdo a la sintaxis :

```
[modificador] class nombreClase [extends superClase] [implements interfaz]
{
  /* Lista de variables y métodos */
}
```

Ejemplo:

```
class primeraClase
{
   ...
 }
```

En este caso, **primeraClase extiende** a la clase **Object**. Es como si, en la primera línea, hubiéramos escrito :

class primeraClase extends java.lang.Object

Cuando una clase es subclase de alguna clase diferente de **Object**, deberá expresarse de manera explícita por medio de la palabra clave **extends**.

Ejemplo :

```
class primeraClase extends claseExistente
{
   ...
}
```

Cada clase deberá almacenarse en un archivo cuya extensión sea **.java** , por lo que a **primeraClase** le correspondería el archivo **primeraClase.java** .

5.3.2.- Definición de métodos.

Los métodos establecen el comportamiento de los objetos, por lo que su definición es esencial. En Java, los métodos son lo que en otros lenguajes se conoce como funciones, procedimientos, o subrutinas por lo que su definición es muy similar.

La sintaxis para definir un método es :

```
[modificador][tipo] nombreDeMétodo ( [tipo arg1,tipo arg2, ..., tipo argN] )
{
  /* Cuerpo del método */
}
```

donde :

tipo es el tipo o clase del valor que regresa el método. Puede ser cualquiera de los tipos básicos (primitivos) o la palabra reservada **void** que indica que el método no regresa valor alguno.

La lista de argumentos que aparece entre paréntesis y entre corchetes es opcional y representan los valores que se pasan al método cuando es invocado. Si el argumento es una variable de alguno de los tipos primitivos, el **paso** se hace **por valor**, lo que significa que al método se le envía una copia del valor almacenado en la variable. En caso de que el argumento sea un objeto, el paso se hace **por referencia**, lo que significa que se pasa la dirección de memoria donde se almacena el objeto y que cualquier cambio realizado en el método (con el nombre de la variable donde se recibe el argumento) afectará al objeto original referenciado.

Ejemplo :

```
class PropiedadEnVenta
{
  boolean credito = true;

  void informa(String direccion, String tipo, float precio )
  {
    if(credito==true)
      System.out.println("Esta propiedad tiene atractivos planes de crédito. \n\n");
    else
      System.out.println("Lo sentimos. Esta propiedad solo se vende al contado.");
    System.out.println("Dirección : "+direccion) ;
    System.out.println("Tipo : "+tipo) ;
    System.out.println("Precio : $ "precio) ;

  }
}
```

En este ejemplo se ha definido la clase propiedadEnVenta , que contiene un método denominado **informa** . Este método no retorna ningún valor, puesto que se utiliza la palabra **void** como tipo de retorno, y recibe tres argumentos que almacena en las variables **direccion, tipo** y **precio**.

dirección almacena una referencia a un objeto de la clase **String**.
tipo almacena una referencia a un objeto de la clase **String**.
precio almacena un valor de tipo **float**.

El código aquí mostrado no es una aplicación completa, pues requiere de una invocación al método **informa( )** perteneciente a un objeto de la clase **propiedadEnVenta**, por ejemplo :

```
objeto.informa( "Balandra # 48", "Colonial", 250000.00F) ;
```

Con esto, se desplegaría :

Esta propiedad tiene atractivos planes de crédito.

Dirección : Balandra #48
Tipo : Colonial
Precio : $ 250000.00

5.3.3.- Clases y métodos abstractos.

Una clase abstracta es aquella que **no puede ser instanciada**, (no pueden crearse objetos de esa clase) y que solo sirve como una plantilla para sus subclases.

En su definición, una clase abstracta se distingue porque su nombre está prefijado por la palabra clave **abstract** y/o porque tiene al menos un método abstracto.

Los métodos abstractos son aquellos cuya declaración está prefijada por la palabra clave **abstract** . Además, los métodos abstractos no cuentan con un cuerpo o definición en la clase donde fueron declarados.

Ejemplo :

```
class abstract Figuras
{
   abstract  double area( ) ;
   abstract  double perimetro( ) ;
}
```

En la clase abstracta **Figuras** se han declarado dos métodos abstractos : **area** y **perimetro**. Ambos regresan valores de tipo **double** a sus invocadores.

Las clases abstractas no pueden utilizarse para crear nuevos objetos ya que sus métodos no son capaces de establecer comportamientos. Esto se debe a que solo existen los nombres de los métodos, pero no sus cuerpos (definiciones).

Para utilizar los métodos de una clase abstracta, es necesario crear una subclase de ella, donde se definan los métodos abstractos.

Por ejemplo, con base a la clase **Figuras** , se pueden definir nuevas clases para diferentes figuras geométricas. Esto se logra utilizando la palabra **extends** , como se muestra en el siguiente código :

```
1 : class Circulo extends Figuras
2 : {
3 :    double radio ;
4 :    protected static final double PI = 3.14159265 ; // Constante
5 :    Circulo( ) { radio = 1.0 } ;
6 :    Circulo( double r) {radio = r } ;
7 :    double area( ) { return PI * radio * radio ; }
8 :    double perimetro( ) { return PI * 2 * radio ; }
9 : }
```

Analicemos cada una de las líneas del código anterior :

En la **línea 1** se establece que la clase **Circulo** extiende a la clase **Figuras**.
En la **línea 3** se define la variable **radio** como del tipo **double**.
En la **línea 4** se define la variable **PI** como del tipo **double** y se le asigna un valor inicial.
En la **línea 5** se define un método constructor para la clase **Circulo** (Obsérvese que el nombre del constructor es el mismo que el de la clase). En este caso, el método no tiene argumentos y el cuerpo solo consta de la instrucción : **radio = 1.0 ;** , encerrada entre llaves.
En la **línea 6** se define otro constructor para la clase, solo que este método recibe un valor de tipo **double** en la variable **r**.

En este punto, conviene observar algunas características de los métodos constructores.

- No regresan valor alguno.
- Pueden sobrecargarse, esto es, definirse con diferentes argumentos.
  Cuando existen varias definiciones de un método, ¿ Como saber cuál código ejecutar cuando se invoca al método ?. El mecanismo consiste en la elaboración de una "firma" para cada una de las definiciones. Dependiendo de la cantidad y los tipos de los argumentos utilizados en la invocación, se podrá determinar cual código ejecutar, a pesar de que los nombres de métodos utilizados en las definiciones sean iguales.

  Por ejemplo, al crear el objeto **miCirculo** ,de la clase **Circulo**, con la instrucción :

  Circulo miCirculo = new Circulo( ) ;

  se utilizará la definición de la línea 5 y el **radio** será igual a 1.0 .

  En cambio, si el objeto se crea con la instrucción :

  Circulo miCirculo = new Circulo(1.5) ;

  será utilizada la definición de la línea 6 y a **radio** se le asignará el valor 1.5 .

  Obsérvese que la creación de un objeto se rige por la sintaxis :

  nombreClase nombreObjeto = new nombreConstructor([argumentos]) ;

  La sobrecarga de métodos no solamente se aplica a los constructores, sino a todos los métodos (salvo las excepciones establecidas por los modificadores que se describen más adelante).

Las **líneas 7 y 8** definen a los métodos **area( )** y **perimetro( )** declarados como abstractos en la clase **Figuras**.

Las reglas correspondientes a la abstracción de clases y métodos se pueden resumir de la siguiente manera :

- Cualquier clase con un método abstracto es automáticamente abstracta ; una clase abstracta debe contener al menos un método abstracto.
- Una clase abstracta no puede utilizarse para crear objetos.
- Una subclase de una clase abstracta puede utilizarse para crear objetos si sobrepone cada uno de los métodos abstractos de la superclase y provee una definición (cuerpo) para cada uno de ellos.
- Si una subclase de una clase abstracta no implementa (define) todos los métodos abstractos que hereda, esa subclase es abstracta.

5.3.4.- Interfaces.

Una interfaz es una clase que contiene declaraciones de métodos pero no las definiciones de ellos.

La declaración de una interfaz se distingue de la de una clase abstracta por el hecho de **no** contener la palabra **class** y porque su nombre está precedido por la palabra clave **interface**. Además, la **clase abstracta puede** tener definidos algunos métodos y la **interfaz no debe** tener métodos definidos.

Por ejemplo, si requerimos desplegar figuras geométricas en pantalla, podemos definir la siguiente interfaz :

```
interface Desplegable
{
   void estableceColor( Color c ) ;
   void establecePosicion( double x, double y ) ;
   void dibuja( DrawWindow d ) ;
}
```

Todos los métodos declarados en una interfaz son implícitamente abstractos, de manera que se puede omitir la palabra **abstract**.

Las interfaces se utilizan para suplir la falta de herencia múltiple en Java, ya que una clase puede tener **solo una superclase** y , al mismo tiempo, puede implementar **varias interfaces** utilizando la palabra **implements**.

Por ejemplo, se puede definir una clase denominada **DibujaCirculo** que extienda la clase **Circulo** y que implemente las interfaces **Desplegable** y **Borrable**.

```
class DibujaCirculo extends Circulo implements Desplegable, Borrable
{
  ...
  ...
}
```

Al igual que las clases, una interfaz puede extender una o más interfaces.

Ejemplo :

```
interface Contador extends Impuestos, Declaraciones, Clientes
{
  ...
  ...
}
```

Una restricción adicional de las interfaces es que no puede definir variables , solamente constantes (utilizando los modificadores **static** y **final**).

Ejemplo :

```
class unaClase { static final int ALFA = 1 ; }  // Una clase con una constante.

interface unaInterfaz { static final BETA = 2 ; } // Una interfaz con una
                                                   // constante.
class otraClase implements unaInterfaz
{
  void funcion( )
  {
    int a = unaClase.ALFA ;  // Debe usarse el nombre unaClase porque
                             // otraClase no la extiende .
    int b = BETA ; // No se requiere el nombre de la interfaz, porque
                   // otraClase la implementa.
  }
}
```

5.3.5.- Modificadores de visibilidad.

Los modificadores de visibilidad especifican la posibilidad de acceder a las clases, métodos y variables de otros paquetes, clases ,y métodos ; como se describe en la Tabla 5.9.

Modificador	Significado
public	Una clase o interfaz **public** es visible en cualquier lugar. Un método o una variable **public** son visibles en cualquier lugar donde su clase es visible.
protected	Las clases no pueden ser **protected**. Un método o una variable **protected** son visibles a través del paquete de su clase, y en cualquier subclase de su clase. Una subclase en un paquete diferente al de la superclase puede acceder a los campos **protected** heredados por sus instancias, pero no puede acceder a esos campos en instancias de la superclase.
private	Las clases no pueden ser **private**. Un método o una variable **private** solamente es visible dentro de su propia clase.
private protected	Una clase no puede ser **private protected**. Una subclase puede acceder a los campos **private protected** heredados por sus instancias, pero no puede acceder a esos campos en instancias de la superclase. Un método o variable **private protected** es visible solamente dentro de su propia clase y dentro de cualquier subclase.
predeterminación	Al no especificarse algún modificador, una clase, interfaz, método, o variable es visible solamente dentro de su paquete.

Tabla 5.9.- Modificadores de visibilidad.

5.3.6.- Otros modificadores.

Además de los modificadores de visibilidad, existen otros modificadores que se utilizan para especificar varios atributos de clases, métodos y variables. Estos modificadores se describen en la tabla 5.10 [FLAND96].

Modificador	Utilizado en :	Significado
abstract	**clase**	**La clase contiene métodos no implementados y no puede instanciarse (utilizarse para crear objetos).**
	interfaz	**Todas las interfaces son abstractas. El modificador es opcional en las declaraciones de interfaces.**
	método	**No se provee el cuerpo (definición, implementación) del método.**
final	**clase**	**La clase no puede extenderse.**
	método	**El método no puede sobreponerse.**
	variable	**El valor de la "variable" no puede cambiar.**
static	**método**	**El método es un "método de clase", es implícitamente final, y puede ser invocado por medio del nombre de la clase.**
	variable	**La variable es una "variable de clase". Existe solo una instancia de la variable. Puede accederse por medio del nombre de la clase.**
synchronized	**método**	**El método hace modificaciones no atómicas a la clase o instancia. Debe asegurarse que dos hilos no puedan modificar a la clase o instancia, al mismo tiempo.** P180
transient	**variable**	**La variable no es parte del estado persistente del objeto. (Este modificador no está implementado)**
volatile	**variable**	**La variable cambia asincrónicamente. El compilador no tratará de salvar su valor en los registros.**

Tabla 5.10.- Otros modificadores.

5.3.7.- Paquetes.

Los paquetes son grupos de clases e interfaces que guardan alguna relación entre si. Las bibliotecas de clase en el Java Developer's Kit, están contenidas dentro del paquete **java** , que a su vez contiene estos otros paquetes :

java.applet

Este pequeño paquete contiene la clase **Applet** que es la superclase de todos los applets. Un applet es un pequeño programa en Java que se empotra en un guión HTML.

Además, este paquete contiene tres interfaces : **AppletContext, AppletStub y AudioClip.**

java.awt

Este paquete toma su nombre de las iniciales de Abstract Windowing Toolkit y contiene todas las clases para el manejo de ventanas y gráficos. Las clases de este paquete pueden agruparse en :

Gráficas Estas clases definen colores, fuentes, imágenes, polígonos, y demás.

Componentes Estas clases son componentes de la interfaz gráfica del usuario(GUI), tales como botones, menús, listas, y cuadros de diálogo.

Manejadores de distribución Estas clases controlan la distribución de componentes dentro de sus objetos contenedores.

Además, este paquete contiene los paquetes : **java.awt.image y java.awt.peer** .

java.io

Este paquete contiene clases que permiten la entrada/salida basada en flujos, y proveen acceso al sistema de archivos.

java.lang

Este paquete provee todas las clases relacionadas con el lenguaje Java, tales como Object, String, Number, Exception, Error, etc.

java.net

Este paquete provee todas las clases que proveen acceso a TCP/IP, enchufes(sockets), direcciones de Internet, y URLs.

java.util

Este paquete provee clases de utilería tales como Hashtable, Vector, Enumeration, Properties, etc.

Para una descripción detallada de estos seis paquetes y las clases contenidas en ellos, se recomienda revisar la referencia [FLAND96] .

5.3.7.1.- Acceso a los paquetes.

Las clases en Java tienen acceso predeterminado a las clases que se encuentran en el paquete **java.lang** . Esto es, las clases de este paquete están a disposición de las nuevas clases creadas por el programador.

Por el contrario, si el programador necesita utilizar alguna de las clases de los otros paquetes, deberá hacer referencias explícitas a dichas clases o deberá escribir una instrucción que le permita importarlas.

Ejemplo :

```
// Referencia explícita con notación de punto
public class MiClase extends java.applet.Applet
{
  ...
}
```

puede escribirse como :

```
import java.applet.Applet ;
// Referencia implícita después de la instrucción import.
public class MiClase extends Applet
{
  ...
}
```

La primera parte del ejemplo hace referencia explícita a la clase **Applet**, por medio de la notación de punto (nombre cualificado).

La segunda parte, primero se "importa" la clase **Applet** y luego se hace referencia a ella de manera directa (nombre abreviado).

Si además de la clase **Applet** se va a hacer referencia a alguna otra clase del paquete **lang**, se puede escribir :

```
import java.lang.* ;
```

con lo que se podrán utilizar los nombres abreviados de todas las clases del paquete.

5.3.8.- Aplicaciones en Java.

A una aplicación que se ejecuta dentro de un guión HTML se le conoce como **applet**.
Este nombre sirve para distinguir a estas aplicaciones de otras llamadas **aplicaciones autónomas**, que se ejecutan desde la línea de órdenes del sistema operativo.

A fin de que puedan ejecutarse los programas para ambos tipos de aplicación, la Maquina Virtual de Java (el intérprete de Java) requiere que se compilen los programas fuente y se obtenga un programa en un código especial llamado bytecode (código de byte), que no tiene instrucciones relacionadas con una plataforma específica (razón por la cual debe interpretarse para su ejecución).

El código fuente para los dos tipos de aplicaciones debe almacenarse en archivos con extensión **.java** , aunque su forma interna y su ejecución son distintos.

5.3.8.1.- Aplicaciones autónomas.

Aunque para la elaboración de los tutoriales que nos ocupan no se requiere de las aplicaciones independientes, en esta sección se describirá brevemente su funcionamiento.

Una aplicación autónoma se ejecuta al invocar a la Máquina Virtual de Java, por ejemplo :

C :\ > java HolaMundo

donde :

java es el nombre de la Máquina Virtual de Java y,

HolaMundo es el nombre de la clase principal que se encuentra en el programa en código de bytes **HolaMundo.class**, obtenido por medio de la compilación del programa fuente **HolaMundo.java** .

Las aplicaciones autónomas deben incluir un método denominado **main( )** , como se muestra en el siguiente ejemplo:

```
class HolaMundo
{
    public static void main ( String args[ ] )
    {
        System.out.println("Hola, Mundo ! !" ;
    }
}
```

Este código debe almacenarse en el archivo de texto **HolaMundo.java**,

5.3.8.2.- Applets.

Un applet es una pequeña aplicación que se ejecuta en un visualizador, por lo que el guión HTML deberá incluir una etiqueta que inicie la ejecución del applet.

Los applets de Java están diseñados para ser independientes de la plataforma, al igual que los guiones HTML. Por esta razón, el visualizador deberá contar con una **Máquina Virtual de Java** que traduzca de código de byte a código de máquina.

Para ejecutar una aplicación dentro de un guión HTML se requiere que éste contenga la etiqueta <APPLET>, como en el siguiente ejemplo :

```
<HTML>
<HEAD>
<TITLE>Mi Primer Applet</TITLE>
</HEAD>
<BODY>
<H3>Este es mi primer applet :</H3>
<P>
<APPLET CODE="HolaMun2.class" WIDTH=100 HEIGHT=50></APPLET>
</BODY>
</HTML>
```

El archivo **HolaMun2.class** es el resultado de compilar el archivo **HolaMun2.java** como se muestra a continuación:

```
C :\> javac HolaMun2.java
```

A continuación se presenta el contenido del archivo **HolaMun2.java** :

```
1 : import java.awt.Graphics ;
2 : public class HolaMun2 extends java.applet.Applet
3 :  {
4 :     public void paint (Graphics g)
5 :       {
6 :          g.drawString("Hola, Mundos !", 1,1) ;
7 :       }
8 :  }
```

En la línea 1 se importa la clase **Graphics**, la cual se encuentra en el paquete **awt**, que a su vez se encuentra en el paquete **java**. Esta línea se incluye para poder crear objetos de la clase **Graphics**, como se observa en la línea 4.

En la línea 2 se deriva, de la clase **Applet** y por medio de la palabra **extends**, la clase **HolaMun2**. Esta derivación debe realizarse para todos los applets.

Observe que el nombre de esta clase coincide con el nombre del archivo (**Holamun2.java**) que la contiene.

En la línea 4 se reescribe el método **paint** que recibe como parámetro una dirección a un objeto de tipo **Graphics,** la cual se almacena en la variable **g**.

Finalmente, en la línea 6 se invoca al método **drawString** del objeto referenciado por **g** para dibujar la cadena **Hola, Mundo !** , en las coordenadas 1 , 1 y con el tipo de letra actual.

La etiqueta **<APPLET>** tiene los siguientes atributos :

CODE = "archivo.class" indica el nombre del archivo de clase donde se almacena el applet compilado.

CODEBASE = ruta especifica la ruta del directorio donde se encuentran almacenados los archivos de clase, cuando estos no se localizan en el directorio actual.

WIDTH = valor indica el ancho (en pixeles) del cuadro donde se desplegará el applet.

HEIGHT = valor indica la altura (en pixeles) del cuadro donde se desplegará el applet.

ALIGN = valor define la alineación que tendrá el applet dentro de la página, donde **valor** puede ser:

TEXTOP alinea la parte superior del applet con la parte superior del texto en la línea.

TOP alinea el applet con el elemento más alto en la línea (applet, imagen o la parte superior del texto).

ABSMIDDLE alinea el centro del applet con el centro del elemento más grande en la línea.

MIDDLE alinea el centro del applet con el centro de la línea base del texto.

BASELINE alinea la parte inferior del applet con la línea base del texto. Es equivalente a **BOTTOM**.

ABSBOTTOM alinea la parte inferior del applet con el elemento más bajo en la línea (línea base del texto, applet o imagen)

HSAPCE = valor controla el espacio horizontal en pixeles entre el applet y su espacio circundante a la izquierda y a la derecha.

VSPACE = valor controla el espacio vertical en pixeles entre el applet y su espacio circundante por arriba y por abajo.

Existe otra etiqueta asociada con los applets que se denomina **<PARAM >** y sirve para pasar parámetros del guión HTML al applet. Los parámetros son constantes contenidas en el guión HTML cuyos valores se pasan a variables del applet.

La etiqueta <PARAM> tiene dos atributos :

NAME = cadena que se refiere al nombre dado a la constante en el guión HTML. Este nombre deberá coincidir exactamente con el nombre utilizado en el applet.

VALUE = cádena que determina el valor del parámetro.

Los parámetros pueden servir para que los usuarios interactúen con un applet, puesto que pueden modificar el guión HTML pero no pueden modificar el archivo de clase.

Por ejemplo : si tenemos el siguiente applet que despliega un nombre

```
import java,awt.Graphics ;
import java,awt.Font ;
import java,awt.Color ;

public class saludo extends java.applet.Applet
{
   Font fuente = new Font("TimesRoman", Font.BOLD, 36 ) ;
   String nombre ;
   public void init( )
   {
      this.nombre = getParameter ( "nombre" ) ;
      if( nombre == null ) name = "desconocido" ;
       nombre = "  Hola " + nombre + " ! " ;
   }
   public void paint( Graphics g )
   {
     g.setFont(fuente) ;
     g.setColor(Color.blue) ;
     g.drawString(nombre, 5, 75) ;
    }
}
```

y al usuario solo le proporcionamos el archivo **saludo.class** ; entonces, él puede escribir el siguiente guión HTML y lo denomina **saludo.html** ,

```
<HTML>
<HEAD>
<TITLE> Ejemplo de paso de par&aacutemetros</TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE = "saludo.class" WIDTH = 500 HEIGHT = 75>
<PARAM NAME = nombre VALUE = "Juan Mares Ruiz">
</APPLET>
</BODY>
</HTML>
```

en el visualizador aparecerá un cuadro de 500 pixeles de ancho por 75 de alto y en él se desplegará :

Hola Juan Mares Ruiz !

Si en el archivo **saludo.html** se elimina la línea que contiene la etiqueta **<PARAM>**, entonces se desplegará :

Hola desconocido !

5.3.8.2.1.- Métodos predeterminados para los applets.

Un applet es un bloque de código Java con una tarea bien definida y se implementa por medio de la creación de una subclase de la clase **Applet**. Esta clase define varios métodos que pueden reescribirse y que controlan el funcionamiento de los applets. A continuación se describen brevemente esos métodos.

init ( )

Realiza la inicialización de un applet. La inicialización se lleva a cabo cada vez que se carga un applet y en ella se incluye la creación de los objetos necesarios, la configuración del estado inicial, la carga de imágenes y/o fuentes de caracteres, y el establecimiento de parámetros.

start ( )

Este método hace que el applet empiece a ejecutar su trabajo. El arranque de un applet ocurre varias veces durante su vida útil, a diferencia de la inicialización que ocurre sólo una vez.

stop ( )

Detiene la ejecución del trabajo del applet. La detención y el arranque se complementan. La detención ocurre cuando se abandona la página que contiene un applet en ejecución

destroy ( )

Libera los recursos ocupados por el applet. Este método permite eliminar cualquier hilo en ejecución o liberar cualquier objeto con referencias activas.

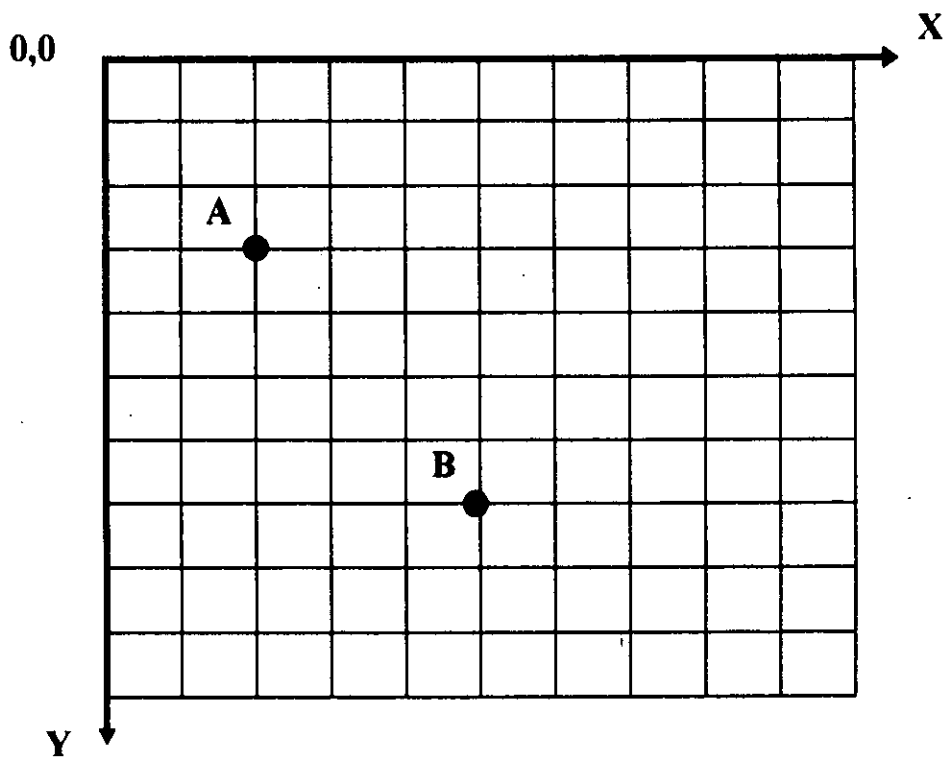
Las implementaciones predeterminadas de estos métodos están vacías, por lo que no realizan tarea alguna. Cuando se requiera que alguno de ellos realice algún trabajo, es necesario reescribirlo.

5.3.8.3.- Gráficos.

Los gráficos son elementos esenciales para describir de manera clara y concisa ciertos conceptos y procesos. En Java, los objetos gráficos se crean por medio de la clase **Graphics**, de la cual describiremos sus principales características. También trataremos con las clases relacionadas **Color** y **Font**.

La clase **Graphics** proporciona un conjunto de métodos para crear algunas figuras geométricas como : líneas rectas, rectángulos, polígonos, óvalos y arcos.

Para dibujar, se utilizan coordenadas **x,y** en el cuarto cuadrante, por lo que los puntos **A(2,3)** y **B(5,7)** se verían así :



También es necesario reescribir el método **paint ( )** , como puede observarse en el siguiente applet de ejemplo :

```
/* Figuras.java : Muestra figuras de ejemplo, dibujadas con métodos de la clase
   Graphics */

import java.awt.* ;

public class Figuras extends java.applet.Applet
{
 public void paint(Graphics g)
 {
   // Dibuja una línea recta desde el punto 10,20 al punto 50,60.
   g.drawLine(10,20,50,60);

   // Dibuja un rectángulo, dadas las coordenadas del vértice
   // superior izquierdo ( 50,60 ), y las dimensiones a= 30 , b=20.
   g.drawRect(50,60,30,20);

   // Dibuja un rectángulo, dadas las coordenadas del vértice
   // superior izquierdo, sus dimensiones a,b y las dimensiones del
   // cuadro de la esquina a redondear.
   g.drawRoundRect(30,90,50,60,20,20) ;

   // Dibuja un rectángulo relleno.
   g.fillRect(130,20,30,40) ;

   // Dibuja un círculo de diámetro = 20 .
   g.drawOval(130,70,20,20);

   // Dibuja un óvalo  diam_x = 50, diam_y = 20.
   g.drawOval(130,100,50,20);

   // Dibuja un arco diam_x = 50, diam_y = 20, ang_ini = 90, ang_fin = 180.
   g.drawArc(130,150,50,20,90,180);
  }
}
```

Al compilarse el archivo **Figuras.java**, se produce el archivo **Figuras.class** , que se utiliza en la etiqueta **<APPLET>** del guión **Figuras.html** mostrado a continuación :

```
<! Figuras.html >
<HTML>
<HEAD>
<TITLE>Dibuja figuras geom&eacutetricas en Java</TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="Figuras.class" WIDTH=400 HEIGHT=200></APPLET>
</BODY>
</HTML>
```

El resultado se muestra en la Figura 5.1

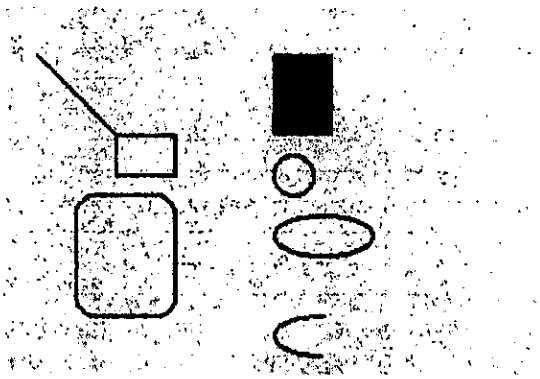


Figura 5.1.- Resultado del guión Figuras.html.

5.3.8.4.- Texto.

Los objetos de la clase **Graphics** también poseen métodos para dibujar caracteres en la pantalla. Antes de dibujar las cadenas de caracteres, deberán crearse objetos de la clase **Font** para indicar la fuente, el estilo y el tamaño de los caracteres a utilizar.

La clase Font se encuentra en el paquete java.awt, y puede importarse como se muestra en el siguiente ejemplo :

```
import java.awt.Font;
import java.awt.Graphics;

public class Fuentes extends java.applet.Applet
{
 public void paint(Graphics g)
 {
  Font p = new Font("TimesRoman",Font.PLAIN,20);
  Font b = new Font("TimesRoman",Font.BOLD,20);
  Font i = new Font("TimesRoman",Font.ITALIC,20);
  Font ib = new Font("TimesRoman",Font.ITALIC+Font.BOLD,20);

  g.setFont(p);
  g.drawString("Esta es una fuente normal", 10, 25);
  g.setFont(b);
  g.drawString("Esta es una fuente en negritas", 10, 50);
  g.setFont(i);
  g.drawString("Esta es una fuente italica", 10, 75);
  g.setFont(ib);
  g.drawString("Esta es una fuente italica negrita", 10, 100);
 }
}
```

El constructor **Font( )** requiere tres argumentos. El primero es el nombre de la fuente, y puede tomar los valores : **"TimesRoman"** , **"Courier"** y **"Helvetica"**.

El segundo es el estilo de fuente representado por el nombre de una de las constantes predefinidas en la clase Font . Los nombres de esas constantes son : **PLAIN** , **BOLD** e **ITALIC**.

El tercer argumento representa el tamaño (en pixeles) con relación a la altura de los caracteres.

El método **drawString( )** , de la clase **Graphics** , dibuja una cadena de caracteres con la fuente establecida por la invocación más reciente al método **setFont( )**.

Los argumentos requeridos por **drawString( )** son : la cadena de caracteres a dibujar, y las coordenadas **x** , **y** que especifican la posición del primer carácter de la cadena.

A continuación se presenta el guión **Fuentes.html** .

```
< ! Fuentes.html >
<HTML>
<HEAD>
<TITLE>Ejemplos de fuentes</TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="Fuentes.class" WIDTH=600 HEIGHT=200></APPLET>
</BODY>
</HTML>
```

El resultado se muestra en la Figura 5.2.

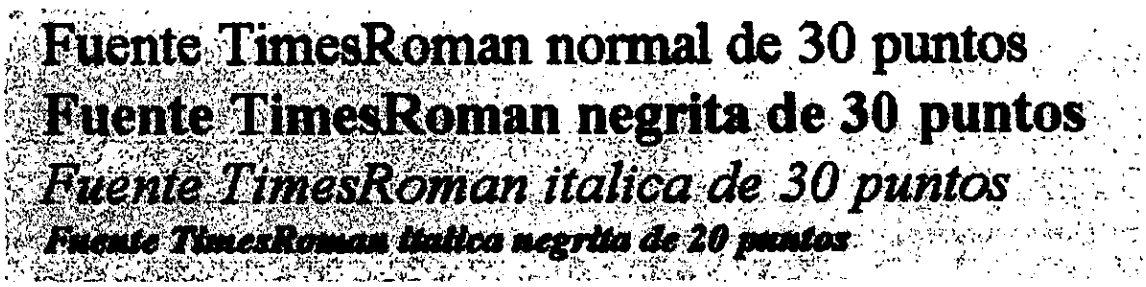


Figura 5.2.- Resultado del guión Fuentes.html.

Existen métodos para buscar información acerca de la fuente utilizada en una cadena, de los cuales se describen algunos en la Tabla 5.11.

Método	Clase	Descripción
getFont( )	**Graphics**	**Regresa el objeto fuente actual .**
getName( )	**Font**	**Regresa el nombre de la fuente**
getSize( )	**Font**	**Regresa el tamaño de la fuente actual en formato de número entero**
getStyle( )	**Font**	**Regresa el estilo actual en formato de número entero (0= normal, 1= negrita, 2= itálica , 3= negrita itálica)**
isPlain( )	**Font**	**Regresa true si el estilo de la fuente es normal**
isBold( )	**Font**	**Regresa true si el estilo de la fuente es negrita**
isItalic( )	**Font**	**Regresa true si el estilo de la fuente es itálica**

Tabla 5.11.- Métodos para el manejo de información acerca de fuentes.

5.3.8.5.- Colores.

En Java, los colores se manejan por medio de los métodos definidos en la clase **Color**.

El constructor **Color( )** describe un color como componentes de los colores rojo, verde y azul (**RGB** ; **R**ed, **G**reen, **B**lue) por medio de valores enteros entre 0 y 255, o valores de punto flotante entre 0.0 y 1.0. De esta manera, el color negro se puede representar como 0,0,0 o como 0.0,0.0,0.0 y el color blanco como 255,255,255 o como 1.0,1.0,1.0

Entre estos valores existen poco mas de 16 millones de combinaciones que posibilitan el manejo de un número igual de colores. Sin embargo, el número de colores que se pueden manejar es de menor cuantía, dependiendo de la plataforma y del visualizador utilizados.

Aunque los colores se pueden representar como ternas de valores, existe un conjunto de nombres de colores predefinidos, como se muestra en la Tabla 5.12.

Nombre :
black
blue
cyan
darkgray
gray
green
lightgray
magenta
orange
pink
red
white
yellow

Tabla 5.12.- Nombres de colores predefinidos en la clase Color.

En el siguiente código se presenta un ejemplo para el uso de nombres de colores predefinidos.

```
/* HolaColores.java : Ejemplo para el manejo de colores predefinidos */

import java.awt.* ;

public class HolaColores extends java.applet.Applet
{
   static final String mensaje = "Hola Colores !" ;
   private Font fuente ;

   // Inicialización por única vez del applet
   public void init( )
   {
      fuente = new Font( "Helvetica", Font.BOLD, 48) ;
   }

  // Dibuja el applet
   public void paint( Graphics g)
   {
      // Dibuja un óvalo rosa.
      g.setColor(Color.pink) ;
      g.fillOval(10,10,330,100) ;

      // Dibuja el contorno rojo, por medio de cuatro óvalos.
      g.setColor(Color.red) ;
      g.drawOval(10,10,330,100) ;
      g.drawOval(9,9,332,102) ;
      g.drawOval(8,8,334,104) ;
      g.drawOval(7,7,336,106) ;

      // Dibuja el texto.
      g.setColor(Color.black) ;
      g.setFont(fuente) ;
      g.drawString(mensaje, 40,75) ;
   }
}
```

El correspondiente guión HTML sería :

```
<! HolaColores.html >
<HTML><HEAD>
<TITLE>Ejemplo para el manejo de colores predefinidos></TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="HolaColores.class" WIDTH=400 HEIGHT=200>
</APPLET>
</BODY>
</HTML>
```

La Figura 5.3 muestra el resultado del guión **HolaColores.html**

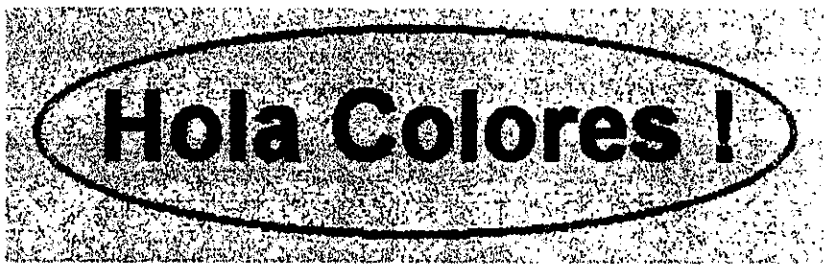


Figura 5.3.- Resultado del guión HolaColores.html

5.3.8.6.- Animación con Java.

La animación en Java consiste de la presentación repetida en pantalla de una secuencia de imágenes, construidas en tiempo de ejecución o previamente elaboradas.

Para esto, es necesario la utilización de hilos (threads) que permitan la ejecución en paralelo de varios procesos. En realidad, un hilo es un espacio de memoria que sirve para controlar un proceso. Con el uso de hilos se garantiza que los recursos del sistema no sean monopolizados y que se puedan tener varios applets en ejecución dentro de la misma página.

Cuando se utilizan hilos es preciso incluir las palabras **implements Runnable** en la identificación de la clase a crear. La interfaz **Runnable** está declarada en el paquete **java.lang**.

Como se muestra en el siguiente ejemplo, deberán reescribirse los métodos **start( )** , **stop( )** y **run( )**.

```
/* Reloj digital */
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;  // Para tomar la fecha del sistema

public class Rdig extends java.applet.Applet   implements Runnable
{
 Font fuente = new Font("TimesRoman",Font.BOLD,24);
 Date fecha;                    // La fecha actual
 Thread ejecutor;               // El hilo para la ejecución del applet

 // Inicia la ejecución del applet
 public void start()
 {
  if (ejecutor == null);        // No está en ejecución el applet
  {
   ejecutor = new Thread(this); // Crea un hilo
   ejecutor.start();            // e inicia la ejecución
  }
 }
```

```
// Detiene la ejecución del applet
 public void stop()
{
  if (ejecutor != null)       // Si está en ejecución el applet
  {
   ejecutor.stop();           // Lo detiene
   ejecutor = null;           // Desactiva el hilo ( el recolector de basura lo recogerá ).
  }
 }

//  Se ejecuta automáticamente cuando se invoca a start( )
 public void run()
 {
  while (true)                // Genera un ciclo infinito
  {
   fecha = new Date();        // Toma la fecha actual
   repaint();                 // Redibuja el cuadro del applet
   try { Thread.sleep(1000); } // Espera 1 segundo para redibujar el cuadro
   catch (InterruptedException e) { } // con un manejo de excepción
  }
 }

//  Dibuja el applet
public void paint(Graphics g)
 {
  g.setFont(fuente);
  g.drawString(fecha.toString(),10,50);  // Convierte la fecha a cadena y la dibuja
 }
} // Fin de la clase  Rdig
```

Este applet se ejecuta con el guión **Rdig.html** mostrado a continuación :

```
<HTML>
<HEAD>
<TITLE>Reloj digital </TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="Rdig.class" WIDTH=600 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

El resultado de visualizar el dibujo del applet en un momento dado, por medio del guión **Rdig1.htm** , se muestra en la Figura 5.4.

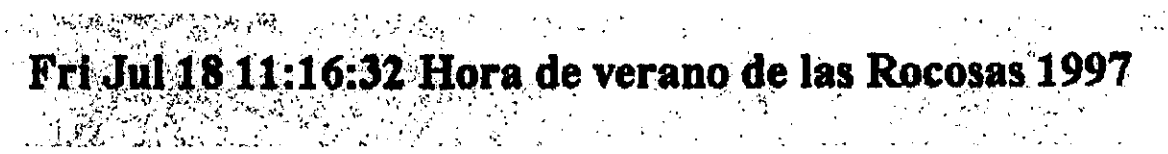


Figura 5.4.- Resultado del guión Rdig.html

El parpadeo que se nota al ejecutar el guión **Rdig.html** se debe a la manera en que funcionan los métodos encargados de dibujar y redibujar los cuadros.

Para generar la animación, es necesario dibujar una imagen, borrar esa imagen, dibujar la siguiente imagen, borrarla, y así sucesivamente.

Para realizar este trabajo se escribe una invocación al método **repaint( )**, el cual primero invoca a **update( )**, que a su vez limpia el cuadro del applet (lo rellena con el color del fondo actual) y enseguida invoca a **paint( )**, quien dibuja el nuevo contenido del cuadro.

El parpadeo se genera por la invocación a **update( )** , quien deja al cuadro "en blanco" por un breve lapso de tiempo.

El parpadeo puede reducirse :

1.- Dibujando sin limpiar el cuadro.

2.- Dibujando solamente las partes del cuadro que sufren algún cambio.

3.- Utilizando doble buffer.

La aplicación de la opción 1 se limita a los casos en que el dibujo no modifica su forma, sino que solamente cambia su color.

La opción 2 implica que el dibujo deberá hacerse por partes, para poder seleccionar aquellas que deberán actualizarse.

En la opción 3 se asigna una área de memoria RAM para utilizarse como almacenamiento "de paso" para las figuras a dibujar. Esta opción es muy adecuada para el caso en que las imágenes que constituyen la animación se encuentran almacenadas en archivos de disco.

5.3.8.6.1.- Animación con imágenes .

Los ejemplos de animación mostrados en la sección anterior han utilizado métodos para el manejo de líneas, fuentes y colores. Esto implica que las figuras utilizadas en las animaciones se tienen que construir cada vez que se dibuja cada cuadro de animación.

Sin embargo, en las animaciones reales generalmente se requieren figuras constituidas por algo mas que líneas, cuadros y óvalos ; se requieren figuras de formas complejas que solo pueden lograrse por medio de paquetes para dibujo o por medio de fotografías. Las imágenes obtenidas por cualquiera de estos dos métodos se almacenan en archivos de disco.

La animación por medio de imágenes almacenadas implica la creación de un guión donde se manifieste la finalidad de la animación, así como la secuencia en que se desplegará cada imagen y la pausa entre cada una de ellas.

Una vez que se tiene el guión, es necesario elaborar cada imagen y almacenarla en un archivo (con extensión .gif o .jpg).

Finalmente, se debe escribir el applet que contenga el código necesario para desplegar la animación.

Aquí nos concentraremos en la descripción de esta última fase y mas adelante veremos un ejemplo completo.

El paquete **java.awt** contiene la clase **Image**, y puede accederse a ella escribiendo la línea :

import java.awt.Image ;

Puesto que las imágenes a utilizar están almacenadas en disco, debe hacerse la copia de cada imagen de disco a memoria. Para esto, se utiliza el método **getImage( )** definido en la clase **Image**.

La copia de la imagen en memoria puede desplegarse por medio del método **drawImage( )** definido en la clase **Graphics**.

A manera de ejemplo, supongamos que tenemos una imagen almacenada en el archivo **unaimag.gif** , y deseamos desplegarla. El código sería :

```
import java.awt.Graphics ;
import java.awt.Image ;

public class Imagen extends java.applet.Applet
{
  Image unaimagen ;
  public void init( )
  {
     unaimagen = getImage ( getCodeBase( ) , "unaimag.gif" ) ;
  }

  public void paint (Graphics g )
  {
     g.drawImage ( unaimagen, 20,20, this ) ;
  }
}
```

Este código solo despliega una imagen, pero sirve de base para la generación de una animación, como se muestra en el siguiente ejemplo :

```
/* ApunAnim.java : Genera una animación */
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Color;
public class ApunAnim extends java.applet.Applet implements Runnable
{
 Image apunimag[] = new Image[7];  // Arreglo de apuntadores a imágenes
 Image imagactual;                 // Imagen actual
 Thread anima;                     // Hilo para manejar la animación.
 int xpos = 1;                     // Coordenada en x ,
 int ypos = 1;                     // Coordenada en y ,
                                   // para la posición de la esquina superior
                                   // izquierda de las imágenes.
 public void init()
 { // Arreglo con los nombres de los archivos de las imágenes
  String archimag[] = { "apunap1.gif",
                        "apunap2.gif",
                        "apunap3.gif",
                        "apunap4.gif",
                        "apunap5.gif",
                        "apunap6.gif",
                        "apunap7.gif", };
  // Crea el arreglo de imágenes.
```

```
 for (int i=0; i < 7; i++)
  {
   apunimag[i] = getImage(getCodeBase(),archimag[i]);
  }
}

public void start()
{
 if (anima == null)
  {
   anima = new Thread(this);
   anima.start();
  }
}

public void stop()
{
 if (anima != null)
  {
   anima.stop();
   anima = null;
  }
}

public void run()
{
 // Inicia la animación.
 setBackground(Color.white);
 ejecutaAnimacion();
}

void ejecutaAnimacion()
{
 for (int i = 0; i < 7; i++)
  {
   imagactual = apunimag[i];
   repaint();
   espera(5000);  // Hace una pausa de 5 segundos
  }
}
```

```
 void espera(int miliseg)
 {
  try { Thread.sleep(miliseg); }
  catch (InterruptedException e) { }
 }

 public void paint(Graphics g)
 {
  g.drawImage(imagactual, xpos, ypos, this);
 }
}
```

El guión HTML correspondiente es :

```
<HTML>
<HEAD>
<TITLE>ApunAnim(Animaci&oacuten con Java)</TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="ApunAnim.class" WIDTH=301 HEIGHT=151></APPLET>
</BODY>
</HTML>
```

La imagen final mostrada por el visualizador es la que se observa en la Figura 5.5.

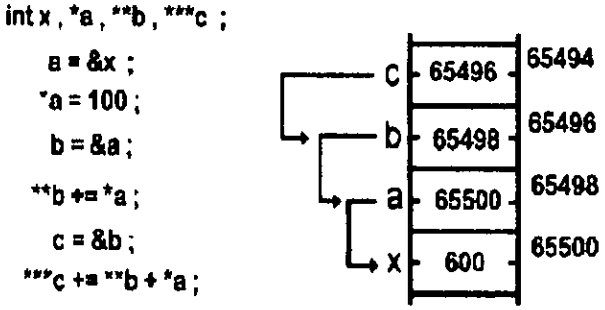


Figura 5.5.- Imagen almacenada en el archivo apunap7.gif

Cuando se ejecuta el applet **ApunAnim**, se observa que el despliegue de las imágenes es irregular. Esto se debe a que el applet empieza a desplegar las imágenes antes de que se hayan cargado completamente desde el disco.

Capítulo 6.- Metodología propuesta.

En este capítulo se proponen una serie de criterios para la elaboración de tutoriales, tomando como caso de estudio el **Tutorial de Lenguaje C++**. A menos que se indique lo contrario, todos los ejemplos de este capítulo se referirán a dicho tutorial.

6.1.- Planificación del tutorial.

La elaboración de un tutorial debe basarse en un plan bien definido, cuya estructura puede variar según los criterios de quien o quienes estén involucrados, pero que siempre deberá definir con claridad cada una de las acciones a seguir.

En nuestro caso, se propone manejar el tutorial como si se tratara de un libro : con una portada, marcas en las hojas que inician cada capítulo, y la posibilidad de avanzar a la siguiente página o retroceder a la página anterior.

Es deseable que en todo momento el usuario del tutorial pueda ubicarse directamente al inicio de cualquier tema. Esto puede lograrse dividiendo la ventana en dos cuadros. En el primero (el de la izquierda) se puede manejar, de manera permanente, el índice que contiene un vínculo para cada uno de los capítulos. El otro cuadro se puede utilizar para desplegar el contenido del tema elegido por el usuario. En la Figura 6.1 se muestra un ejemplo donde el 17% del ancho de la pantalla se asigna al cuadro del índice temático y el 83 % restante al cuadro de contenidos.

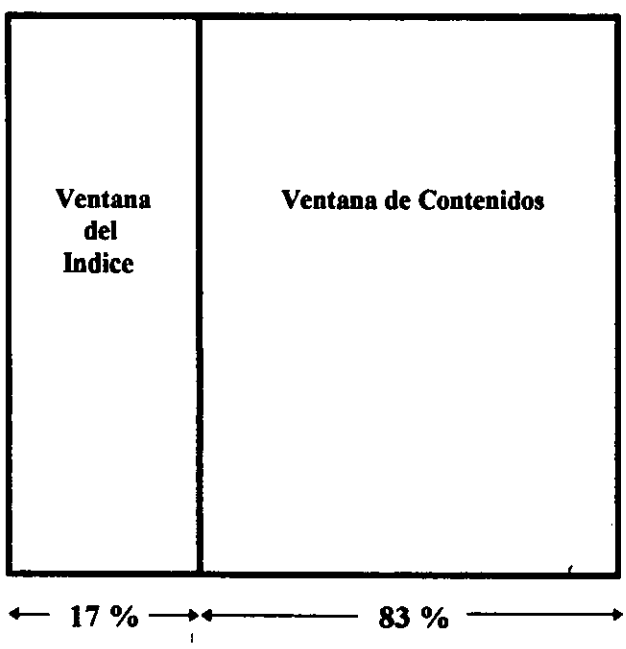


Figura 6.1.- Distribución del índice y el contenido en las páginas del tutorial.

Los porcentajes de distribución solo obedecen a un criterio de preferencias personales, en el que se consideró el tamaño de letra mínimo necesario para leer sin esforzarse demasiado. Se recomienda utilizar proporciones desde 15% , 85% hasta 25% ,75% .

El siguiente código es el contenido del archivo **index.htm**, donde puede observarse la utilización de cuadros.

```
1 : <HTML>
2 : <HEAD><TITLE>Tutorial C++</TITLE></HEAD>
3 : <FRAMESET COLS="17%,83%" BORDER=0>
4 : <FRAME SRC="barra.htm" SCROLLING=no>
5 : <FRAME SRC="portada.htm" NAME="principal">
6 : </FRAMESET >
7 : </HTML>
```

En la línea 3 (los números se agregaron para efecto de este análisis) se divide la ventana actual en dos cuadros columna. El cuadro de la izquierda tomará el 17% del ancho de la ventana y el cuadro de la derecha tomará el 83% restante.

En la línea 4 se asigna al cuadro de la izquierda el contenido del archivo **barra.htm**, indicándose que este cuadro no tendrá barra de desplazamiento (Por omisión, los cuadros poseerán una barra de desplazamiento).

En la línea 5 vemos que al cuadro de la derecha se le asigna el contenido del archivo **portada.htm**. A este cuadro se le asigna el nombre **principal**, para referencias posteriores.

El resultado de visualizar el guión **index.htm** se muestra parcialmente en la Figura 6.2.

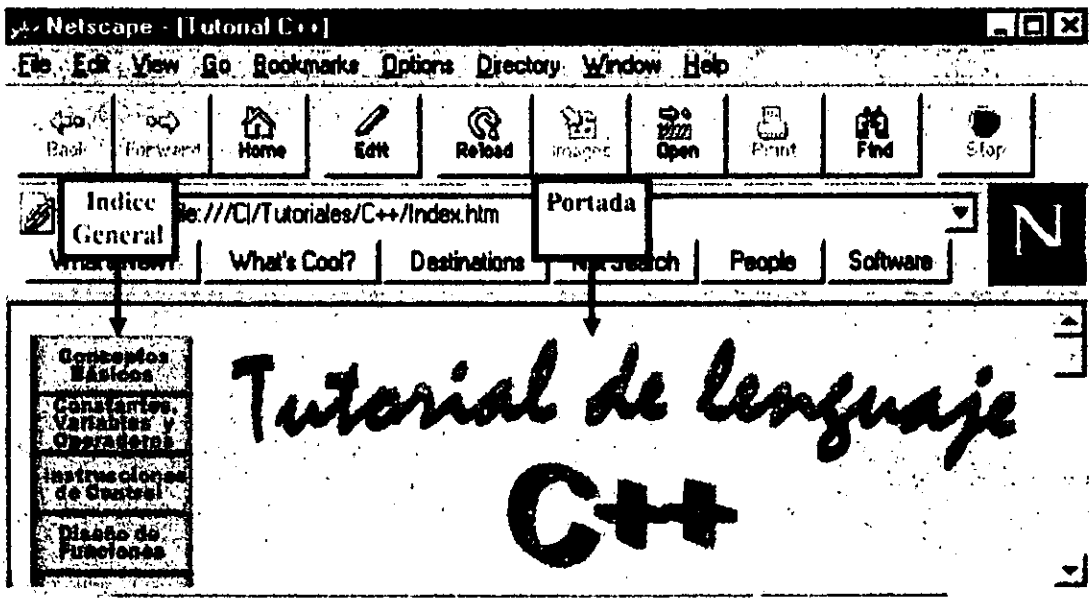


Figura 6.2.- Pagina base del Tutorial de Lenguaje C++.

La estructura jerárquica del tutorial se diseñó tomando en cuenta la página base, páginas de segundo nivel que representan los índices de los capítulos y las páginas de contenidos temáticos, como puede observarse en la Figura 6.3 .

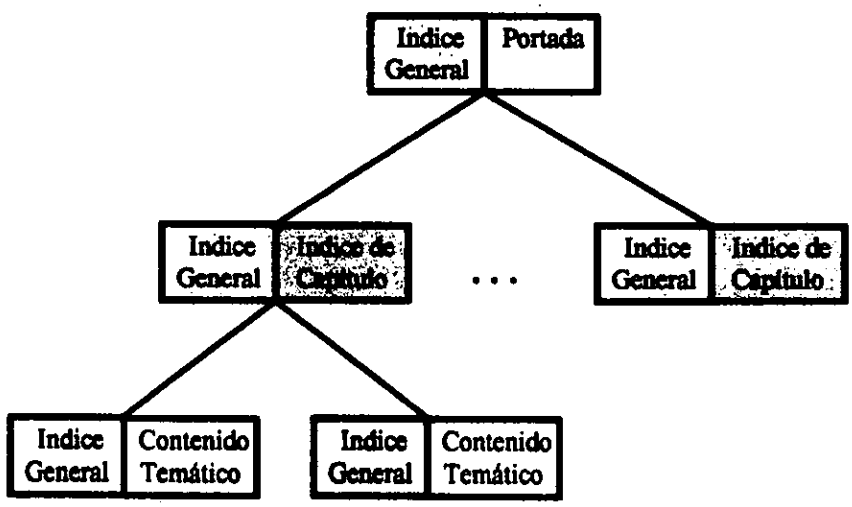


Figura 6.3.- Estructura jerárquica del sistema Tutorial de Lenguaje C++.

La página base se representa en la Figura 6.2 , donde se pueden distinguir el índice general y la portada.

Las páginas de segundo nivel corresponden a los índices de cada capítulo, como se muestra en la Figura 6.4.

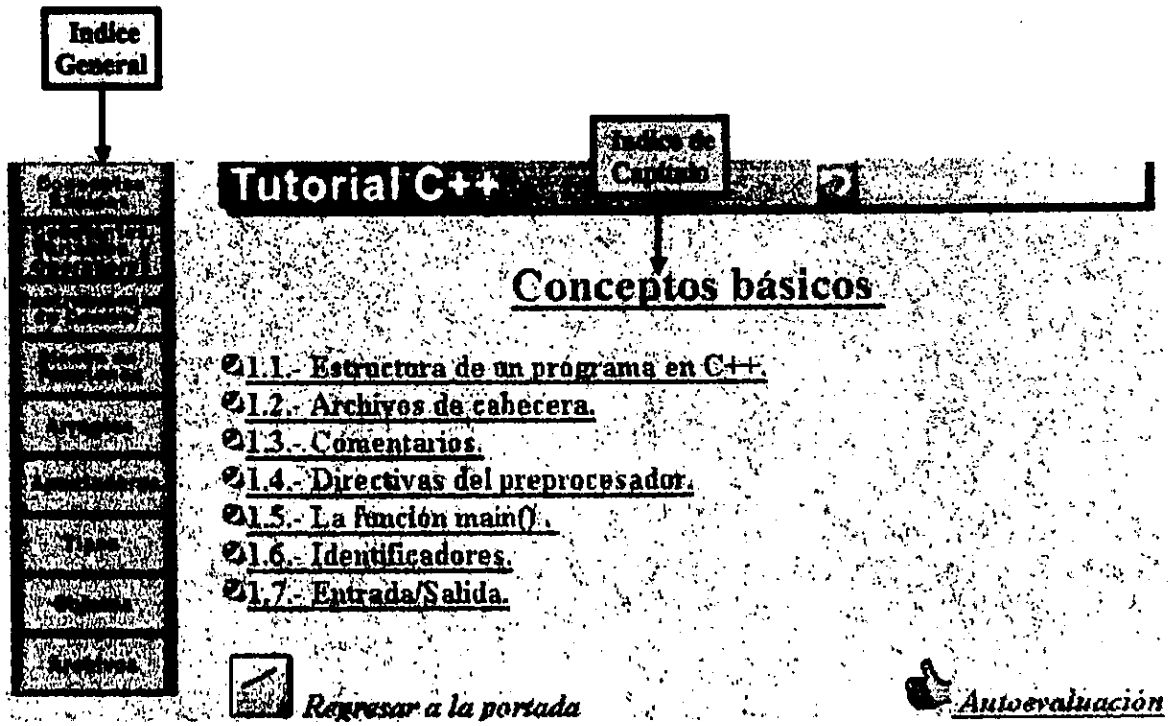


Figura 6.4.- Ejemplo de una página de segundo nivel
(Indice del Capítulo 1.- Conceptos Básicos)

Las páginas de contenido temático se utilizan para manejar los elementos que representan la esencia del tutorial en sí, como son: el texto, los gráficos, las animaciones, etc. En la Figura 6.5 se muestra un ejemplo de página de contenido temático.

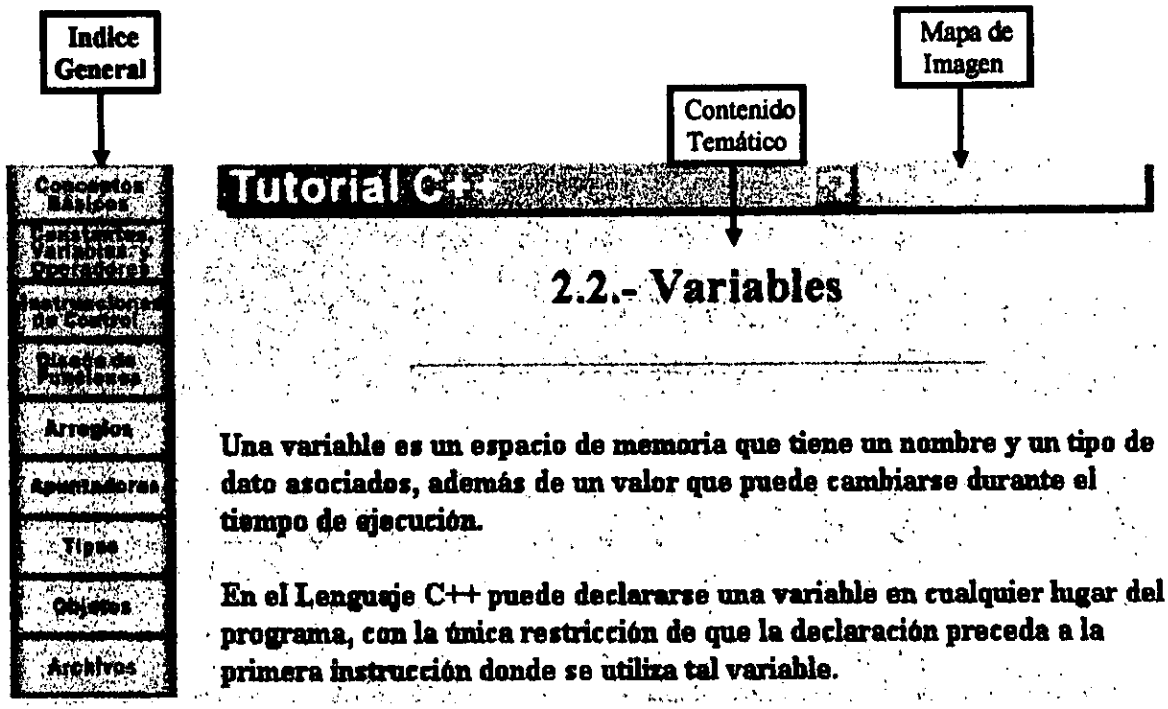


Figura 6.5.- Ejemplo de una página de contenido temático.

Aunque la estructura del tutorial es jerárquica, también se pueden visitar las páginas de manera secuencial desde la primera hasta la última, como si se tratara de las páginas de un libro. Para esto se agregan, al final de cada página de contenido temático, imágenes que permiten vincular hacia la página anterior y hacia la página siguiente, como puede observarse en la Figura 6.6.

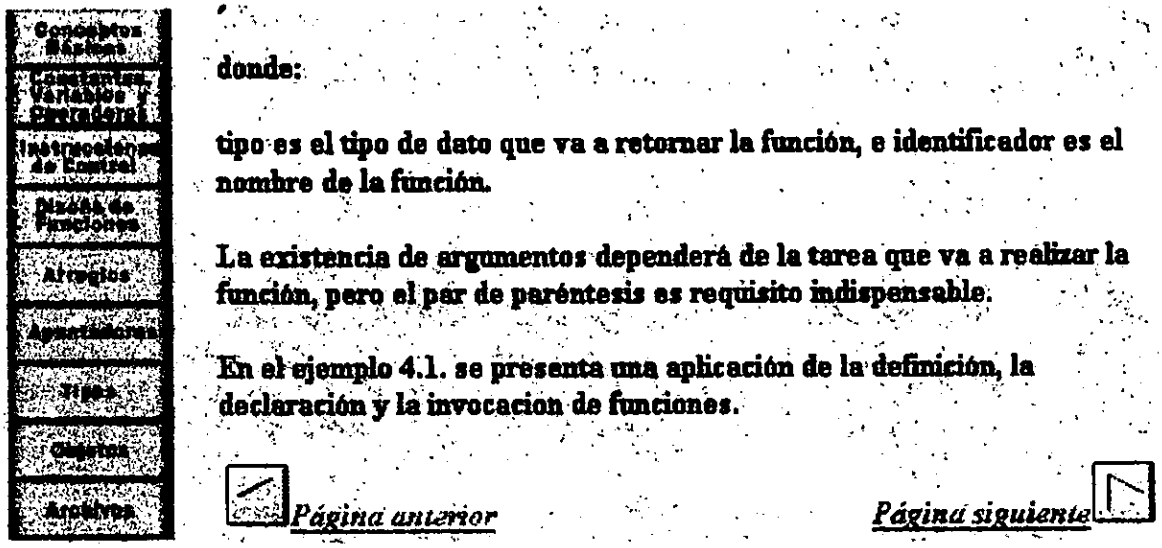


Figura 6.6.- Ejemplo de vinculación entre páginas del tutorial.

Otro aspecto que debe cuidarse, con respecto a la navegación por las páginas de un tutorial, es la facilidad con que el usuario pueda regresar a la página base. Para esto, en el Tutorial de Lenguaje C++ se ha incluido el gráfico señalado en la parte superior de la Figura 6.5, el cual contiene un **mapa de imagen** llamado **linea** que permite vincular con la página **portada.htm** , con lo que se logra la misma apariencia que se observa al ejecutar el archivo **index.htm**, al inicio de la sesión.

6.2.- Preparación del texto.

La esencia de un tutorial está constituida por el texto que representa una serie de ideas y conceptos que se quieren compartir. El texto debe prepararse por medio de un editor que permita escribir caracteres especiales (tales como las letras vocales acentuadas).

Una vez que se tiene escrito el texto con caracteres especiales, sustituir aquellos que sean considerados como reservados por el lenguaje HTML.

Existen programas convertidores que se pueden utilizar para convertir el contenido de archivos elaborados con algún editor de textos popular. Es muy probable que, aún con el uso de un convertidor, se tenga que escribir directamente algo de texto HTML.

Algunos visualizadores ya cuentan con un editor integrado, de manera que no se tenga necesidad de salir de su ambiente para modificar el texto de documentos HTML.

En el caso del Tutorial de Lenguaje C++, se tomaron como base las Notas de Lenguaje C++ que yo había preparado en WordPerfect. Antes de convertir las notas a guiones HTML, primero fueron convertidas a documento de Microsoft Word.

Una vez que se tienen elaborado el texto y los gráficos (o al menos definidos los nombres de éstos), se pueden insertar las etiquetas HTML que definirán la apariencia de las páginas que conforman el tutorial.

La inserción de etiquetas es una actividad manual que deberá realizarse aún en el caso de que se esté utilizando una herramienta para convertir el texto a guión HTML, ya que es común que se requieran ciertas etiquetas que las herramientas de traducción actuales no detectan (como es el caso del texto preformateado controlado por las etiquetas <PRE> y </PRE>)

6.3.- Uso de gráficos.

Debido a que el uso de Internet es intensivo, conviene que el número de gráficos mantenga en el menor nivel sin menoscabo en la calidad mínima requerida para la presentación del tutorial. De igual manera, se recomienda que los archivos que almacenan los gráficos se reduzcan al mínimo tamaño posible. De esta manera se disminuirá el riesgo de que el tutorial no sea utilizado a causa del exceso de tiempo requerido para transportar los archivos a través de la red.

Los archivos de gráficos a utilizar en Internet deben ser de formato **GIF** o de formato **JPEG**.

El formato de archivo **GIF** es un formato de 8 bits y es propiedad de **CompuServe** ; pero el algoritmo de compresión que utilizan pertenece a **Unisys**.

El formato de archivo **JPEG** fué desarrollado por el Joint Photographic Experts Group (de cuyas iniciales toma su nombre) y , como no está patentado, su uso no requiere de consideraciones legales. Puesto que sirve para el manejo de gráficos con calidad fotográfica, el formato **JPEG** es de 24 bits .

Debido a su formato de 8 bits, las imágenes **GIF** pueden manejar como máximo 256 colores, mientras que la imágenes **JPEG** pueden manejar un poco más de 16 millones.

Se recomienda crear gráficos con el mínimo número de colores necesario, ya que el número de colores con que se crea un gráfico determina el tamaño del archivo que lo contiene.

Dentro de las ventajas que ofrece el uso de archivos con formato **GIF** pueden mencionarse las siguientes :

- Están soportados por todos los visualizadores de Web .
- Permiten la inclusión de varias imágenes dentro de un solo archivo.
- Pueden proporcionar animación, al manejar dentro de un ciclo las imágenes del archivo.

Las imágenes o gráficos se pueden utilizar tanto para diseñar el fondo de las páginas como para generar vínculos, como se observa en el guión **portada.html** mostrado a continuación.

```
1 : <HTML>
2 : <HEAD>
3 : <TITLE> Portada</TITLE>
4 : </HEAD>
5 : <BODY BACKGROUND="bk103.jpg" text="#394A64">
6 : <CENTER><IMG SRC="tuto1.gif"></CENTER>
7 : <CENTER><IMG SRC="logo1.gif"></CENTER><br>
8 : <CENTER><A HREF="acercade.htm" onMouseOver='status="Acerca de" ;
9 :  return(true)' onMouseOut='status="Document: Done"'>
10 :  <IMG border=0  SRC="acercade.gif">
11 : </A></CENTER>
12 : </BODY>
13 : </HTML>
```

Como puede observarse en la línea 5 del código anterior , uno de los atributos de la etiqueta **<BODY>** es **BACKGROUND** el cual tiene un valor igual a **bk103.jpg**. Esto significa que el fondo de la página deberá rellenarse con el gráfico contenido en el archivo **bk103.jpg**. La extensión **.jpg** indica que el archivo tiene un formato **JPEG**.

En la línea 6, la etiqueta **<IMG>** tiene un atributo **SRC="tuto1.gif"**, con lo que deberá desplegarse la imagen contenida en el archivo **tuto1.gif**. La línea 7 produce un efecto similar.
La etiqueta **<A>** de la línea 8 significa que se va a manejar un vínculo. El atributo **HREF** especifica que se va a establecer un vínculo hacia el archivo **acercade.htm**. En la línea 10 se especifica que el vínculo va a estar formado por la imagen contenida en el archivo **acercade.gif**.

La elaboración de gráficos puede realizarse de manera simultánea o alternada con la elaboración del texto. Dependiendo del número de personas que estén involucradas en el proyecto, se pueden formar grupos especializados en gráficos.

Para el Tutorial de Lenguaje C++, rescatamos (por medio de Word) algunos de los gráficos que contenían las notas. Los gráficos que no pudieron rescatarse, o que no cumplieron con los requerimientos de calidad, se volvieron a elaborar utilizando el paquete Corel Draw. Por medio de Corel Draw, se crearon gráficos que se exportaron como archivos JPG (a 16 millones de colores). Estos archivos JPG se filtraron, se redujeron de tamaño y se grabaron como archivos GIF a 256 colores (con PaintShop Pro 4.1), para que ocuparan menos espacio y se transmitieran más rápidamente por Internet.

En algunos casos se grabaron las imágenes en archivos GIF de 2 y 16 colores para reducir el espacio de almacenamiento necesario. En otros casos, dados los requerimientos de calidad de los gráficos, se grabaron en archivos JPG a 16 millones de colores.

6.4.- Autoevaluaciones de tutoriales escritas en JavaScript.

Las autoevaluaciones son herramientas que permiten al usuario de un tutorial revisar su grado de avance en el conocimiento de los temas revisados. Por otra parte, estas herramientas constituyen factores de gran motivación para la mayoría de las personas puesto que, además de constituir un reto, les permite ubicar su nivel de conocimiento acerca del tema. Además, al resolver un cuestionario de autoevaluación no se corre ningún riesgo puesto que puede realizarse cuantas veces se quiera en completa soledad frente a la computadora y no afectará nuestro estatus en el grupo (pérdida de calificación en el curso o baja estima de los compañeros).

Para la elaboración de cuestionarios de autoevaluación, se recomienda lo siguiente :

- Cada una de las autoevaluaciones deberán corresponder a un bloque bien definido de conocimiento, como son los temas unidades o capítulos. Esto a fin de que el aprendiz pueda centrar su atención en pocos conceptos íntimamente relacionados .
- Las preguntas de una autoevaluación deben ser precisas y únicas.
- Siempre deberá informarse al usuario si la respuesta que eligió fue la correcta o no.
- En caso de respuesta incorrecta deberá retroalimentarse al usuario desplegándole la respuesta correcta.
- Llevar la cuenta de las respuestas correctas y hacer un resumen al final para **que el usuario determine** si la proporción de respuestas correctas es suficiente para pasar a otro tema.
- Deben evitarse mensajes o sonidos "graciosos" que pudieran desalentar al aprendiz o afectar su autoestima. Aunque no sirve de base para una calificación oficial, la autoevaluación debe tratarse con la seriedad que merece alguien que se esfuerza en adquirir nuevos conocimientos.

En el siguiente listado se muestra, a manera de ejemplo, el código contenido en el archivo **indice1.htm** (adaptado de [TESST296]). correspondiente al **Tutorial de Lenguaje C++** manejado como caso de estudio en este trabajo. Este código sirve para controlar la autoevaluación de la unidad 1.

```
1 : <!-- Contenido del archivo indice1.htm -->
2 : <HTML>
3 : <SCRIPT LANGUAGE="JavaScript">
4 : <!-- Oculta de los antiguos navegadores de NetScape --
5 : var totalnum=5 ;
6 : var correctans=0 ;
7 : var count=0 ;
8 : var arraycount=1 ;
9 : var totalright=0 ;
10 : var correcttext="blank" ;
11 : //-- Finaliza el ocultamiento -->
12 : </SCRIPT>
13 : <FRAMESET ROWS="*,0">
14 : <FRAME SRC="autoeva1.htm">
15 : <FRAME SCROLLING="no" NORESIZE>
16 : </FRAMESET>
17 : </HTML>
```

Desde la línea 5 hasta la 10 se declaran las siguientes variables :

totalnum número total de preguntas en la autoevaluación. El número de preguntas se fija con un valor igual a 5.

correctans índice del elemento del arreglo que contiene la respuesta correcta. Toma valores de 0 a N-1.

count contador para el número de preguntas realizadas.

arraycount índice para controlar el arreglo donde se guardan las preguntas y respuestas. Se inicializa a 1 porque es el subíndice de la primera cadena del arreglo.

totalright número total de preguntas contestadas correctamente.

correcttext texto de la respuesta correcta.

Todas estas variables serán consideradas como globales, ya que se definen en el cuadro "padre". Los cuadros "hijo" tendrán acceso irrestricto a estas variables en cualquier momento, como veremos más adelante en el listado de **autoeva1.htm**.

La línea 13 contiene la etiqueta de HTML :

<FRAMESET ROWS="*,0">

Esta etiqueta establece que la ventana actual (donde se está ejecutando el guión HTML) va a dividirse en dos cuadros. Al segundo cuadro le asigna 0 renglones de los disponibles, y al primer cuadro le asigna el resto de los renglones disponibles (todos).
Esto debe hacerse para que el visualizador inicie la actualización de un cuadro.

En la línea 14 :

<FRAME SRC="autoeva1.htm">

Se indica que el primer cuadro se va a utilizar como ventana para el guión almacenado en **autoeva1.htm**.

En la línea 15 se establece que el contenido del segundo cuadro no podrá desplazarse **(SCROLLING="no")**. El atributo **NORESIZE** indica que dicho cuadro no podrá redimensionarse al ejecutarse el guión.

En el listado siguiente se presenta el contenido del archivo **autoeva1.htm**, que corresponde al código utilizado para la auto evaluación de la unidad 1 en el tutorial de lenguaje C++.

```
 1 : <!-- Contenido del archivo autoeva1.htm -->
 2 : <BODY background="bk103.jpg">
 3 : <SCRIPT LANGUAGE="JavaScript">
 4 : <!--Oculta de los antiguos visualizadores --
 5 : var totalans=3  // Variable del cuadro local, que representa el número de respuestas
                        por cada pregunta.
 6 : function MakeArray(n)  // Define un objeto arreglo.
 7 : {
 8 :      this.length=n  //  A la primera propiedad ( length, con índice cero), se le asigna el
                          número de elementos del arreglo.
 9 :      for(var i=1;i<=n;i++) // Las propiedades restantes,
10 :        {
11 :            this[i]=0          // se inicializan a cero todos los atributos del objeto.
12 :        }
13 :        return this
14 : }
```

```
15 : function checkout(questionnum) // Controla la salida de la auto evaluación y/o
                                        de cada pregunta.
16 : {
17 :      if(parent.count > parent.totalnum) // Si el usuario hace click en un botón de radio
                                              después de que terminó la prueba.
18 :       {
19 :          alert('Para regresar, haga click en "Regresar a la unidad 1."')
20 :       }
21 :      else
22 :       {
23 :          if(questionnum==parent.correctans) // Si la respuesta seleccionada
                                                    actualmente es la correcta.
24 :           {
25 :              parent.totalright++
26 :              alert("\n Correcto...")
27 :           }
28 :          else  // Si la respuesta es incorrecta, despliega la correcta.
29 :           {
30 :              alert("\nIncorrecto... La respuesta es:\n\n"+parent.correcttext)
31 :           }
32 :          parent.count++  // Incrementa el contador para ir a la siguiente pregunta.
33 :          if(parent.count==parent.totalnum) // Si count = número total de preguntas
34 :           {
35 :              parent.count++ // Incrementa de nuevo el contador para que, si es
36 :           }                 //  mayor que totalnum, active la alerta si trata de hacer
37 :       }                     //  click en un botón después de finalizada la prueba.
38 :     location.href=location.href // Carga de nuevo la página actual, conservando los
39 : }                             // valores de las variables globales.
40 : questans=new MakeArray(eval((parent.totalnum+1)*(totalans+2)));
41 : questans[1]="1.- El lenguaje C++ es un: "
42 : questans[2]="subconjunto del lenguaje C."
43 : questans[3]="superconjunto del lenguaje C."
44 : questans[4]="lenguaje de bajo nivel."
45 : questans[5]=1 // La respuesta correcta es: "un superconjunto del lenguaje C."
46 : questans[6]="2.- El procesamiento de macros se realiza por: "
47 : questans[7]="el compilador."
48 : questans[8]="el enlazador."
49 : questans[9]="el preprocesador."
50 : questans[10]=2 // La respuesta correcta es: "el preprocesador."
51 : questans[11]="3.- El prototipo de la funci&oacuten printf() se define en: "
52 : questans[12]="el archivo de cabecera stdio.h"
53 : questans[13]="el kernel del sistema operativo."
54 : questans[14]="el archivo de cabecera iostream.h"
55 : questans[15]=0// La respuesta correcta es: "el archivo de cabecera stdio.h"
```

```
56 : questans[16]="4.- Los identificadores en C++: "
57 : questans[17]="deben iniciar con una letra min&uacutescula"
58 : questans[18]="pueden iniciar con cualquier letra o el caracter de subrayado"
59 : questans[19]="no pueden tener m&aacutes de 8 caracteres"
60 : questans[20]=1// La respuesta correcta es: "pueden iniciar con cualquier letra o el
                                                        caracter de subrayado"
61 : questans[21]="5.- Para manejar el objeto cin"
62 : questans[22]="debe incluirse el archivo stdio.h"
63 : questans[23]="es indistinto incluir stdio.h  o iostream.h"
64 : questans[24]="debe incluirse iostream.h"
65 : questans[25]=2// La respuesta correcta es: "debe incluirse iostream.h"
66 : questans[26]="Fin de la prueba. Gracias"
67 : questans[27]=""
68 : questans[28]=""
69 : questans[29]=""
70 : questans[30]=255 // Respuesta correcta: ninguna.
71 : if(parent.count>parent.totalnum)  // Si el usuario contestó todas las preguntas.
72 :  {
73 :       document.write("<H2>La prueba termin&oacute. Usted tuvo "
74 :       +parent.totalright+" aciertos de un total de "+parent.totalnum
75 :       +" preguntas. </H2>")
76 :   }
77 : else  // Si todavía quedan preguntas.
78 :  {
79 :       arrayind=parent.arraycount
80 :       document.write('<FORM METHOD="POST">')
81 :       document.write("<H2>", questans[arrayind++],"</H2>")
82 :       document.write('<LI><INPUT TYPE="radio" onClick="checkout(0)">')
83 :       if(questans[arrayind+3]==0)
84 :         {
85 :            parent.correcttext=questans[arrayind]
86 :         }
87 :       document.write(questans[arrayind++],"<BR>")
88 :       document.write('<LI><INPUT TYPE="radio" onClick="checkout(1)">')
89 :       if(questans[arrayind+2]==1)
90 :         {
91 :            parent.correcttext=questans[arrayind]
92 :         }
93 :       document.write(questans[arrayind++],"<BR>")
94 :       document.write('<LI><INPUT TYPE="radio" onClick="checkout(2)">')
95 :       if(questans[arrayind+1]==2)
96 :         {
97 :           parent.correcttext=questans[arrayind]
98 :         }
```

```
99 :       document.write(questans[arrayind++],"<BR>")
100 :      document.write("</FORM>")
101 :       parent.correctans=questans[arrayind++]
102 :       parent.arraycount=arrayind
103 :       document.write("<BR><BR>")
104 :       document.write("Usted ha completado "+parent.count+" de "
105 :       +parent.totalnum+" preguntas, con "+parent.totalright)
106 :       if(parent.totalright=="1")
107 :         {
108 :             document.write(" respuesta correcta.")
109 :         }
110 :       else
111 :         {
112 :             document.write(" respuestas correctas.")
113 :         }
114 :   }
115 : document.write("<BR><BR>")
116 : document.write('<LI><A HREF="unidad1.htm"><B> Regresar a la unidad 1</A>')
117 : //-- Finaliza el ocultamiento -->
118 : </SCRIPT>
119 : </BODY>
```

Algunas de las líneas del listado anterior requieren comentarse con mayor detalle.

En la línea 40, el código:

questans = new MakeArray(eval((parent.totalnum+1)*(totalans+2)));

questans representa una nueva instancia creada por medio de MakeArray.

parent.totalnum+1 La variable totalnum pertenece al cuadro "padre" y corresponde al número total de preguntas (igual a 5, en nuestro caso). El número de preguntas se incrementa en 1 debido a que debe considerarse la inclusión de la pregunta : **"Fin de la prueba. Gracias".**

totalans+2 establece que el número de respuestas para cada pregunta deberá incrementarse en 2, ya que en el arreglo questans, por cada pregunta se tiene el siguiente formato :

questans[1] = Pregunta1
questans[2] = Respuesta0
questans[3] = Respuesta1
questans[4] = Respuesta2

```
questans[5] = RespuestaCorrecta1

questans[6] = Pregunta2
questans[7] = Respuesta0
questans[8] = Respuesta1
questans[9] = Respuesta2
questans[10] = RespuestaCorrecta2
...
```

Se utiliza el método **eval** para calcular el número de elementos del arreglo **questans**.

La cantidad de elementos de está dado por el número de preguntas de la evaluación, multiplicado por el número de preguntas/respuestas de cada pregunta.

En nuestro caso, esto es : **(5+1)* (3+2)** = 30

Las líneas que contienen la invocación al método **write** del objeto **document** (tales como la línea 80), sirven para escribir texto en el documento que se maneja en el cuadro actual (el cuadro "hijo", en este caso). De esta forma se controla la aparición de ciertos párrafos en un documento HTML.

En la línea 80 se escribe en el documento actual la siguiente etiqueta de apertura:

<FORM METHOD="POST">

La etiqueta de apertura <FORM> se utiliza para indicar que el contenido que sigue es un formulario y, por lo general, tiene dos atributos : METHOD y ACTION. Cuando se requiera utilizar el valor POST, deberá incluirse el atributo METHOD (tal como se hizo aquí) ya que su valor predeterminado es GET .

GET y POST se denominan así por los comandos HTTP que usa el visualizador para comunicarse con el servidor. Conviene utilizar POST para que el sistema no fije límites al número de caracteres que puedan contener las variables a utilizar [LEMAL96].

En la línea 81 se despliega la siguiente pregunta, utilizando el tamaño de letra correspondiente a un encabezado de nivel 2.

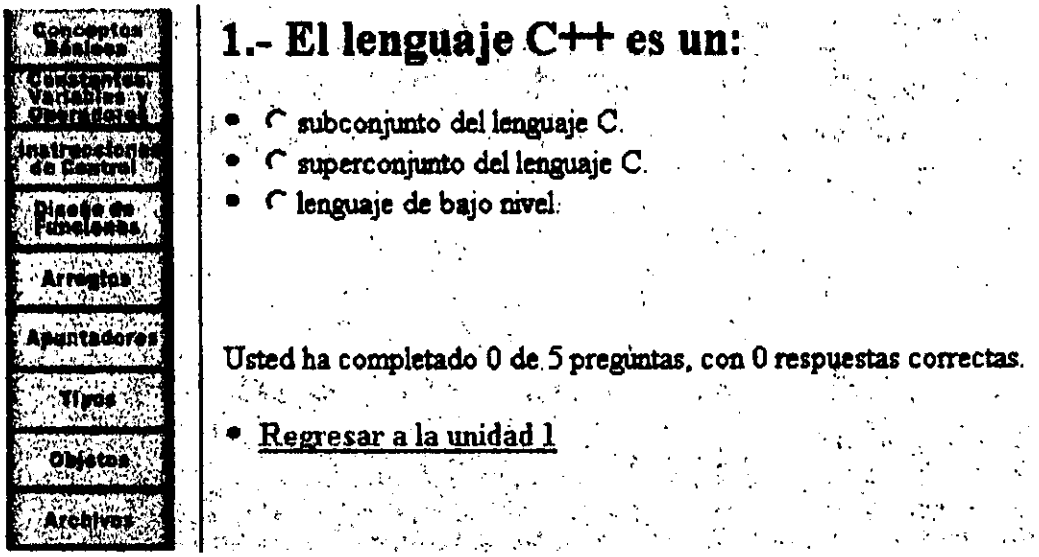
En la línea 82 se manejará como un elemento de lista (<LI>) todo lo que sigue, hasta encontrar otra etiqueta <LI>.

La etiqueta <INPUT> indica un elemento del formulario. El valor del atributo TYPE indica que la clase de entrada va a ser un botón de radio. En el caso del atributo **onClick**, su valor **checkout(0)** establece que, cuando se haga un clic sobre el botón de radio, se invocará al método **checkout()**, pasándole como parámetro el valor 0. Este valor indica que la pregunta a procesar será la primera.

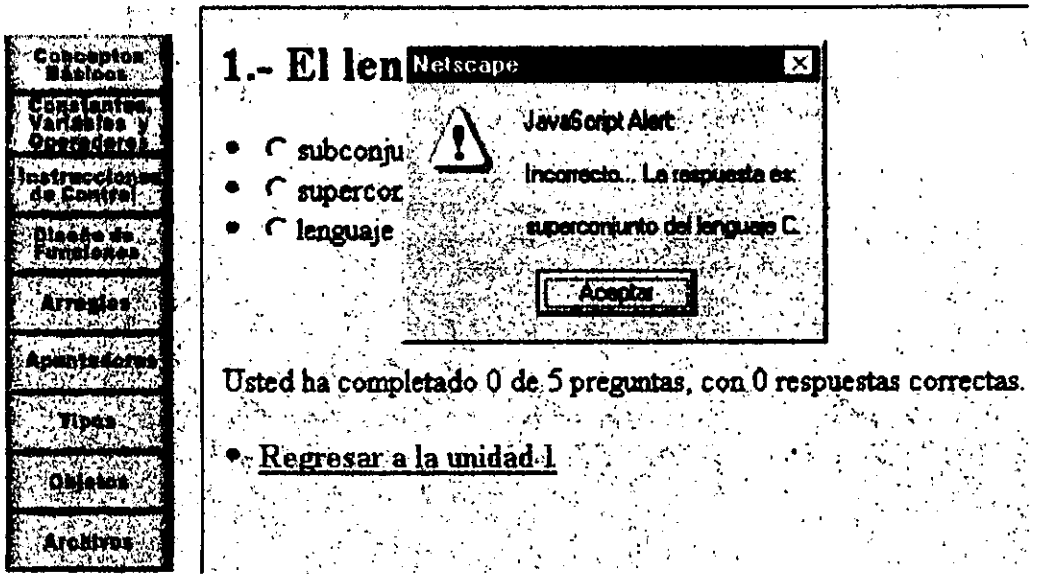
En la línea 83 se revisa si la respuesta correcta para esta pregunta es la respuesta 1 y, de ser así, en la línea 85 se asigna el texto de la respuesta correcta a la variable **correcttext** del cuadro "padre".

De las líneas 87 a la 97 se realiza un proceso similar, pero ahora para las respuestas 2 y 3.

Por ejemplo, si hacemos clic en el icono de la autoevaluación de la Unidad 1 (Conceptos básicos) del **Tutorial de Lenguaje C++**, el visualizador despliega:



Si, en esta ventana, el usuario elige la primera de las tres respuestas posibles, aparecerá:



Cuando el usuario haga clic sobre el botón **Aceptar** aparecerá la pregunta 2, y así sucesivamente hasta completar las 5 preguntas de la autoevaluación

6.5.- Animación.

La animación consiste de la presentación de una serie de imágenes en secuencia, de manera que den la impresión de movimiento.

En la creación de tutoriales, para ciertos temas se requiere la inclusión de animaciones que muestren los pasos a seguir durante la realización de un proceso complejo o que simulen el funcionamiento de cierto mecanismo.

Como recurso didáctico, la animación puede ser muy útil pero hay que dosificarla de manera adecuada, ya que puede resultar muy cara en términos del tiempo necesario para transportar las imágenes a través de la red y del tiempo requerido por el procesador para producir el efecto de movimiento.

Por principio, se debe evitar que la animación se realice en el servidor puesto que éste ya tiene bastante trabajo atendiendo los requerimientos de los clientes. Conviene realizar la mayor parte posible del trabajo de procesamiento en el lado del cliente, de manera que el trabajo del servidor se limite al envío de los archivos de documentos e imágenes.

En el Tutorial de Lenguaje C++, en la sección **6.10.- Apuntadores a apuntadores**, se presentan dos ejemplos de animación: uno con un gif animado y otro con un applet animado.

Al pulsar el botón izquierdo del ratón sobre el icono del gif animado se despliega una ventana como la que mostrada en la Figura 6.7.

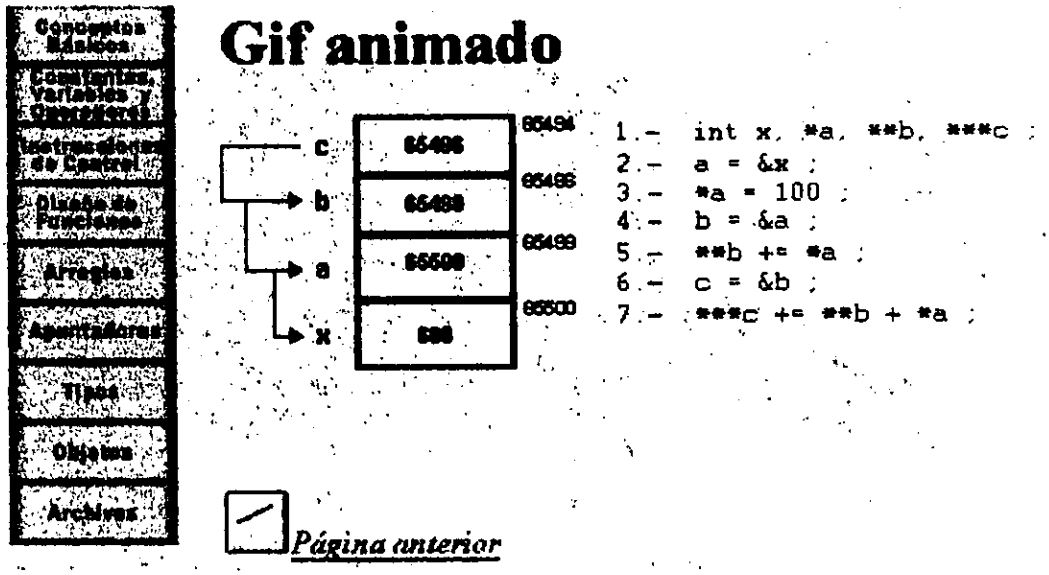


Figura 6.7.- Ejemplo de gif animado.

Al pulsar el botón izquierdo del ratón sobre el icono del applet animado se despliega una ventana como la que mostrada en la Figura 6.8.

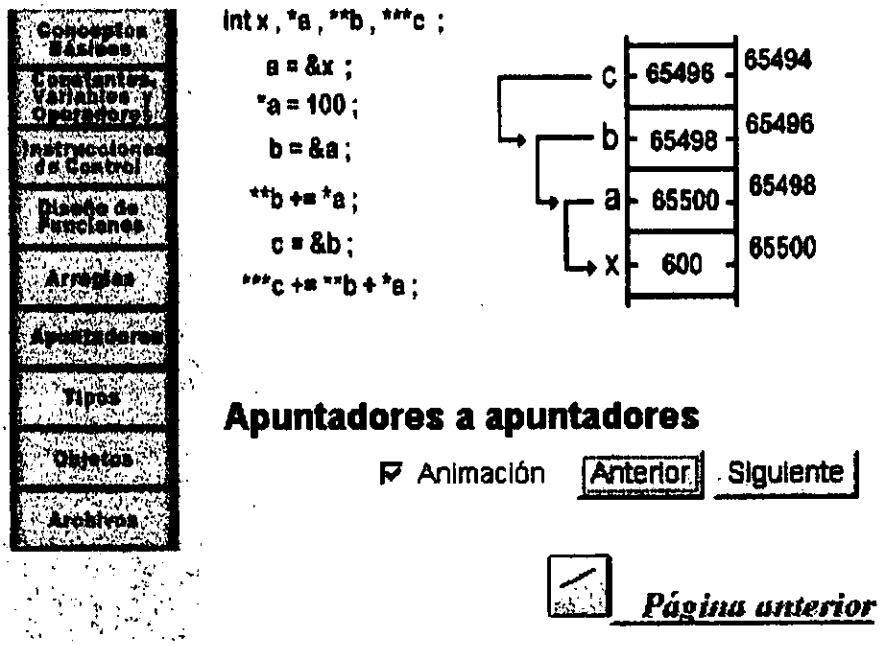


Figura 6.8.- Ejemplo de applet animado.

6.6.- Interactividad.

La interactividad permite al usuario intervenir (generalmente por medio del teclado y del ratón) para modificar el comportamiento de un programa.

En el lenguaje de programación Java, además de la posibilidad de personalizar el comportamiento de un applet utilizando parámetros, existe un paquete denominado **awt** (AWT Abstract Windowing Toolkit) que proporciona, entre otros componentes : ventanas, menús, botones, casillas de verificación, campos de texto, barras de desplazamiento y listas desplazables.

El siguiente código corresponde al applet animado mostrado en la sección anterior (6.5), y en él se utilizan algunos componentes que incrementan la interactividad. Para asegurar que la carga de las imágenes se realice antes de que se inicie la ejecución del applet, se utilizará un objeto de la clase **MediaTRacker.**

```
/* ApunAni2.java : Genera una animación mejorada e interactiva  */

import java.awt.*;
import java.net.URL;

public class ApunAni2 extends java.applet.Applet implements Runnable
{
 MediaTracker carga;
 Image apunimag[] = new Image[7];
 Thread anima;
 int indice=0 ;
 Button avance, retroceso ; // Declara dos botones.
 Checkbox auto; // Casilla de verificación para la animación.
 Label titulo;  // Título para la animación.
 Panel marquesina, control ; // En la marquesina va el título y el control.

 public void init()
 {
  carga = new MediaTracker(this);

  String archimag[] = { "apunap1.gif",
                        "apunap2.gif",
                        "apunap3.gif",
                        "apunap4.gif",
                        "apunap5.gif",
                        "apunap6.gif",
                        "apunap7.gif", };
```

```
// Crea el arreglo de imágenes.

for (int i=0; i < 7; i++)
 {
  apunimag[i] = getImage(getCodeBase(),archimag[i]);
 }

try{
   carga.waitForID(0);
  }
catch(InterruptedException e)
  {
   System.out.println("Falla al cargar imagen...");
  }
repaint();
carga.checkID(1,true);

// Crea el diseño de la pantalla

setLayout(new BorderLayout()); // Ventana de primer nivel.

avance = new Button("Siguiente");
retroceso = new Button("Anterior");
auto = new Checkbox("Animación");
auto.setState(true);
titulo = new Label("Apuntadores a apuntadores");

Panel marquesina = new Panel();
setBackground(Color.white);
marquesina.setLayout(new BorderLayout()); // Ventana de segundo nivel.
marquesina.add("North", titulo); // Pone el título en la parte superior
                        // de la marquesina.
Panel control = new Panel();
control.setLayout(new FlowLayout()); // Ventana de tercer nivel.
control.add(auto);
control.add(retroceso);
control.add(avance);

setFont(new Font("Helvetica", Font.BOLD, 18));
add("South",marquesina);
setFont(new Font("Helvetica", Font.PLAIN, 14));
marquesina.add("South", control);

}
```

```
// Monitorea las acciones de casillas de verificación y botones.

public boolean action(Event evento, Object x)
{
 if(evento.target == retroceso)
   {
    if(indice!=0)
      {
       indice--;
       repaint();
      }
    return true;
   }
 else if(evento.target == avance)
   {
    indice++;
    repaint();
    return true;
   }
 else
   return false;
}

public void start()
{
  repaint();
  anima = new Thread(this);
  anima.start();
}

public void stop()
{
 repaint();
 if (anima != null)
   {
    anima.stop();
    anima = null;
   }
}
```

```
public void run()
{
 Thread corrida = Thread.currentThread();
 setBackground(Color.white);
 while(anima == corrida)
  {
   try{
       Thread.sleep(5000);
     }
   catch(InterruptedException e)
     {
      break;
     }
   if(auto.getState() == true)
    {
     if(carga.checkID(1,true))
      synchronized(this)
       {
        if(indice < 0) indice=0;
        else indice++;
       }
     repaint();
    }
  }
}

public void update(Graphics g)
{
 try{
     paint(g);
   }
 catch(ArrayIndexOutOfBoundsException e)
   {
    if(indice < 0) indice = 0;
    else indice = 7;
    System.out.println("No hay m s im genes...");
   }
}
```

```
 public void paint(Graphics g)
 {
  if(carga.isErrorAny())
    {
     g.setColor(Color.black);
     g.fillRect(0,0,size().width, size().height);
     return;
    }
  g.drawImage(apunimag[indice], 0, 0, this);
 }
}
```

El correspondiente guión HTML es :

```
<HTML>
<HEAD>
<TITLE>ApunAni2(Animaci&oacuten con Java, versi&oacuten 2)</TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="ApunAni2.class" WIDTH=400 HEIGHT=250></APPLET>
</BODY>
</HTML>
```

El resultado desplegado por el visualizador se muestra en la Figura 6.8 desplegada en la sección 6.5.

6.7.- Pruebas.

Deberá probarse el material que resulte de los pasos anteriores y realizar modificaciones cuantas veces sea necesario. Conviene tener en mente que ya existen etiquetas HTML en el texto a modificar(sobre todo cuando se piensa en realizar sustituciones globales de texto).

6.8.- Instalación.

Todo el trabajo mencionado en los puntos anteriores se puede llevar a cabo en una máquina que no se encuentre conectada a una red. Sin embargo, la intención es compartir los tutoriales a través de Internet, por lo que debemos instalar nuestrá aplicación en un servidor Web.

Un servidor Web es un programa que se encuentra en una máquina de Internet esperando a que un visualizador Web se conecte con él y haga una solicitud. Una vez que llega la solicitud por el cable, el servidor localiza el archivo y lo envía al visualizador.

Debido a que los servidores se comunican por medio de hipertexto, en ocasiones se les llama servidores HTTPD. La D se debe a que los servidores Web se comportan como los programas Unix llamados daemons (demonios), que se ejecutan en el fondo y se encuentran siempre a la espera de solicitudes.

Existen tres formas posibles para instalar una aplicación en un servidor Web:

A).- Utilizando el servidor Web de la Organización.

Si la empresa o institución donde usted trabaja cuenta con un servidor Web, la instalación de su aplicación se puede lograr con la asesoría del personal encargado de la administración del sistema.

B).- Contratando la publicación de una página .

Un proveedor de servicio Internet (ISP) es una compañía que proporciona acceso a Internet. Por lo general, estos proveedores cuentan con varios servidores y un enlace de alta velocidad hacia la dorsal de Internet. El acceso se logra a través de un módem, la línea telefónica y la utilización de un protocolo de línea (tal como el protocolo punto a punto **ppp**). Los proveedores cobran una cuota fija mensual por determinada cantidad de tiempo y un cargo extra por el tiempo excedente. Con frecuencia, los proveedores también ofrecen el servicio de publicación de páginas Web y rentan espacio del disco de su servidor con lo que se tiene un **anfitrión Web**.

Contratar la publicación de su página puede ser una opción adecuada si :

- no tiene una gran cantidad de información que publicar,
- no piensa cambiar con mucha frecuencia los contenidos,
- no le preocupa que, en ciertos periodos, sus clientes tengan acceso lento o limitado al sitio,
- no tenga planes para manejar una base de datos de tamaño considerable, ni para interactuar por correo electrónico con los clientes que visiten su página, y
- le interese hacer una prueba por un tiempo limitado y a un costo bajo.

C).- Instalando un sitio Web .

Si usted necesita publicar muchas páginas o incluir en ellas cualquier tipo y cantidad de contenido (incluyendo guiones CGI y mapas de imágenes) , sin estar sujeto a las restricciones de un anfitrión Web, entonces la alternativa es instalar su propio **sitio Web**.

Un sitio Web está formado por una computadora, un sistema operativo, un programa servidor Web, contenidos que puedan ver quienes visiten el sitio, y una conexión a Internet.

Para que el servidor pueda atender de manera eficiente a muchos visitantes, es necesario que el sitio Web cuente con una computadora cuyas características de velocidad de procesamiento y capacidad de almacenamiento sean las mejores posibles, de acuerdo al presupuesto disponible.

Las tecnologías de los procesadores y los dispositivos de almacenamiento están evolucionando con gran rapidez, por lo que cualquier configuración se vuelve obsoleta en poco tiempo (entre 3 y 5 años). Por esta razón, deben tomarse con reserva los modelos que se recomiendan a continuación [ZIMMSC96]

Configuración de equipo.

- **Tarjeta principal** : con un procesador Pentium con la mayor velocidad de reloj posible (de 100 MHZ en adelante).
- **Gabinete** : de torre para una mayor facilidad de instalación de los dispositivos periféricos.
- **Tarjetas controladoras** : para vídeo, UART 16550 en el puerto serial para módem externo, para red (en su caso), y para disco.
- **Fuente de poder** : de 250 watts, para suministrar la energía requerida por 4 unidades de disco duro, unidad para discos flexibles, unidad para CD-ROM, equipo de respaldo, tarjetas adaptadoras.
- **Memoria RAM** : de 16 MB en adelante, para mayor velocidad de procesamiento.

- **Monitor** : super VGA de 15 o 17 ". Debido a que el equipo para el sitio Web necesita estar permanentemente encendido, conviene uno de los que reducen el consumo de energía.

- **Unidad de disco duro C** : de tamaño suficiente para servir como disco de arranque. Por conveniencia de precio por megabyte, debe elegirse el tamaño menor que se fabrique en el momento de la adquisición. Una alternativa es utilizar algún disco disponible cuya capacidad sea de al menos 512 MB.
- **Unidad de disco duro D** : de al menos 1 Gygabyte de capacidad. Conviene aplicar el criterio de comprar el disco de mayor capacidad permitido por el presupuesto, para asegurar una mayor vida útil. Otro criterio puede ser el de costo por megabyte.
- **Unidad CD-ROM** : de la mayor velocidad disponible (8x o 16x). Estas unidades se han vuelto necesarias debido a que muchos paquetes de software se distribuyen en CDs.
- **Unidad para discos flexibles de 3.5"** : para manejar archivos de menos de 1.44 MB y mantener la compatibilidad con sistemas que no cuentan con otro medio para el intercambio de archivos.
- **Dispositivo de respaldo** : con capacidad de almacenamiento igual o mayor que el disco duro de mayor capacidad con que cuente el sistema. Puede consistir de una unidad de cintas o de discos removibles.
- **Fuente de poder ininterrumpible** ; indispensable para alcanzar a cerrar los sistemas sin que se sufran pérdidas de información al interrumpirse el suministro habitual de energía.
- **Módem V.34 o una interfaz ISDN** (Integrated Services Digital Network , Red Digital de Servicios Integrados).

Además del equipo mencionado anteriormente, para que el sitio Web funcione se requiere de una :

Conexión a Internet.

Para que un proveedor brinde servicio para un sitio Web, se requiere :

1).- Una cuenta Internet PPP (Point to Point Protocol, Protocolo Punto a Punto) de acceso telefónico para negocios, de al menos 28.8 Kbps (kilo bits por segundo).

2).- Una línea para módem y un puerto en la máquina del proveedor.

Un sitio Web debe correr las 24 horas del día, por lo que se debe contar con una **conexión de acceso telefónico** (también denominada **en demanda**) o una **línea dedicada** (también denominada **exclusiva**). La diferencia es que la línea de acceso telefónico se conecta a una cuenta mancomunada (pool) para módem en las instalaciones del proveedor, mientras que la línea dedicada proporciona un número telefónico privado hacia el proveedor. Las ventajas de ésta última consisten en que :

- No existe el problema de la señal de ocupado.
- Cuando se pierde la conexión, se puede regresar inmediatamente a la línea marcando al proveedor en el número privado.

3).- Una dirección IP estática.

4).- Un registro de DNS (Domain Name Service, Servicio de Nombre de Dominio), por ejemplo : **mi_empresa.com**

Puesto que cada una de las interfaces en Internet debe tener una dirección IP única, se debe registrar el nombre de dominio ante el Centro de Información sobre la Red Internet (Inter NIC), a través de un formulario que se puede conseguir a través del proveedor o en **http://rs.internic.net/rs-internic.html**.
Una vez que se ha llenado el formulario, se envía por correo electrónico a **hostmaster@internic.net**.

5).- Un registro CNAME (Canonic NAME, Nombre Canónico) para **www.mi_empresa.com** . Los nombres canónicos son nombres alternativos para el dominio que permiten utilizar, por ejemplo, el URL **http ://www.mi_empresa.com/** para el sitio Web .

6).- Verificar cuáles servidores de protocolos puede soportar el proveedor, por ejemplo :

- **HTTP** - Protocolo de transferencia de Hipertexto.
- **FTP** - Protocolo de Transferencia de Archivos.
- **SMTP** - Protocolo Simple de Transferencia de Correo.

La instalación del Tutorial de Lenguaje C++ se hizo en un servidor de la Universidad Autónoma de Baja California Sur, donde hasta la fecha tenemos publicados los tutoriales descritos en la siguiente tabla :

Tutorial	URL
Lenguaje C++	**http ://www.uabcs.mx/teclapaz/tutorc/index.htm**
Estructura de Datos	**http ://www.uabcs.mx/teclapaz/estru1/index.htm**
Teleproceso	**http ://www.uabcs.mx/teclapaz/telepro/index.htm**

Además, hemos recibido el apoyo para la publicación del presente trabajo en :

http ://www.uabcs.mx/teclapaz/guiatuto/index.htm

y en

http ://pitahaya.uabcs.mx/spa/dsc/jac/guiatuto/index.htm

Conclusiones y recomendaciones.

Se espera que los tutoriales elaborados con la metodología aquí propuesta sirvan para rescatar, en el corto plazo y con adaptaciones menores, el material que utilizan los profesores en su trabajo diario. El medio en que se distribuirán los tutoriales permite un considerable ahorro de tiempo y de recursos de impresión comparado con la publicación tradicional de libros y revistas en papel, además de que se lograrán mejoras en el desempeño del material disponible, puesto que se podrán incluir características de animación e interactividad, agregando, además de los lenguajes utilizados en este trabajo, el lenguaje VRML (Virtual Reality Markup Language , Lenguaje de Marcado de Realidad Virtual) para incluir simulaciones tridimensionales en los tutoriales.

Considerando las limitaciones de instalaciones y equipo que enfrentan la mayoría de las instituciones educativas, así como el tipo de material escrito que los profesores pueden aportar en el corto plazo, se recomienda que los tutoriales a elaborar inicialmente sean del tipo CAI . Conforme se disponga de un mayor ancho de banda y de equipo mas avanzado, se podrán aumentar el tamaño de los gráficos y los audioclips, incrementando las capacidades multimedia de los tutoriales.

Esto no quiere decir que solo se recomienda mejorar el aspecto cosmético de los tutoriales. La presentación es muy importante para captar la atención de los usuarios, pero también es importante que los tutoriales evolucionen acercándose lo mas posible al nivel de los tutoriales inteligentes.

Para obtener los mejores resultados, es conveniente que se establezcan primero las políticas de diseño generales y luego se aplique el proceso de diseño sobre un prototipo que permita afinar los criterios y motive la colaboración del grupo de trabajo, teniendo siempre en mente que la interacción grupal es determinante para el desarrollo de la capacidad creativa de los individuos.

Es muy importante que todos los integrantes del grupo colaboren en y estén enterados de la evolución del prototipo, aún en el caso de que se decida formar subgrupos por áreas de interés (redacción de contenidos, elaboración de programas, elaboración de gráficos y animación, etc..)

En este trabajo se ha propuesto una manera de aprovechar esa gigantesca red que es Internet para mejorar el proceso enseñanza-aprendizaje, de manera que se facilite el trabajo de profesores y alumnos. Si, además, los tutoriales pueden ser de utilidad a otras personas que no estén relacionadas con el sistema educativo que los elaboró, entonces se habrán rebasado con creces las metas originales.

Bibliografía.

[BARTN96] Barttlet Neil, Leslie Alex, Simkin Steve ; " Java Programming Explorer " ; Coriolis Group Books ; USA ; 1996.

[BERNE94] Berners-Lee T., Conolly D. ; "HyperText Markup Language Specification - 2.0" ; HyperText Markup Language Working Group, Internet Draft; 1994.

[DANEA96] Arman Danesh ; "Aprendiendo JavaScript en una semana" ; Prentice Hall Latinoamericana, S.A. ; México ; 1996.

[DECKR93] Decker Rick, Hirshfield Stuart ; "The Object Concept : An Introduction to Computer Programming Using C++" ; PWS Publishing Company ; USA ; 1993.

[FISHEE93] Fishetti E. , Gisolfi A ; " Desde la Instrucción Asistida por Computadora Hasta los Sistemas Tutoriales Inteligentes " ; Revista Sistemas, Depto. de Sist. Comp., UABCS, Agosto 1993; México ; 1993.

[FLAND96] Flanagan David ; " Java in a Nutshell" ; O'Reilly & Associates,Inc ; USA ; 1996.

[GULBD96] Gulbrasen David, Rawlings Kenrick ; "Cree sus Applets Para WEB con Java" ; Prentice Hall Latinoamericana, S.A. ; México ; 1996.

[KROLE94] Krol Ed ; " Conéctate al mundo de Internet" Segunda Edición ; McGraw-Hill Interamericana de México, S.A. de C.V. ; México ; 1994.

[LEMAL96] Lemay Laura ; "Aprendiendo HTML 3.0 para Web en una semana" ; Prentice Hall Latinoamericana, S.A. ; México ; 1996.

[LEMAP96] Lemay Laura, Perkins Charles L. ; "Aprendiendo Java en 21 días" ; Prentice Hall Latinoamericana, S.A. ; México ; 1996.

[LAURA92] Laureano Cruces Ana Lilia C. ; " MICRARMS : un Sistema de Enseñanza Basado en el Conocimiento de Análisis de Armaduras "; Tesis de Maestría; UNAM ; México ; 1992.

[PARKB96] Park Woolf Beverly ; " Intelligent Multimedia Tutoring Systems " ; Communications of the ACM, April 1996, p. 30 ; USA ; 1996.

[PARKT95] Park Woolf Beverly ; " Aprendiendo TCP/IP en 14 días " ; Prentice Hall Hispanoamericana, S.A. ; México ; 1995.

[OBREA89] Obregón Sánchez Arturo ; " EDA : Un sistema tutorial inteligente en el área de estructuras de datos " ; Tesis de Maestría ; UNAM ; México ; 1989.

[RICHE94] Elaine Rich, Kevin Night ; " Inteligencia Artificial " ; McGraw-Hill/Interamericana de España, S.A. ; España ; 1994.

[SLEED82] Sleeman D., Brown J.S. ; " Intelligent Tutoring Systems " ; Academic Press ; London ; 1982.

[TESST196] Tessier Tom ; " Using JavaScript to Create Interactive Web Pages "; Dr. Dobb's Journal, March 1996, p. 84 ; USA ; 1996.

[TESST296] Tessier Tom ; " Sharing Data Between Web Page Frames Using JavaScript "; Dr. Dobb's Journal, May 1996, p. 72 ; USA ; 1996.

[VANHA96] Van Hoff Arthur, Shaio Sami, Starbuck Orca ; " Hooked on Java " ; Addison-Wesley Publishing Company ; USA ; 1996.

[ZIMMSC96] Zimmerman Scott, Brown L.T. ; " Kit de Construcción de Sitios Web para Windows 95 " ; Prentice Hall Hispanoamericana, S.A. ; México ; 1996.