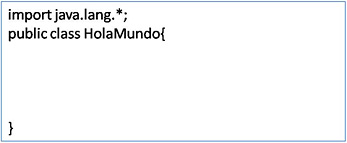
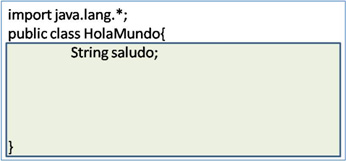
FUNDACION UNIVERSITARIA SAN MARTIN, FUSM

PROGRAMACION ORIENTADA A INTERNET

Ruben Dario Alvino Sanabria, Cod, 112504, Ing. sistemas

*Introduccion:* Para comprender mejor la sintaxis del programa debemos pensar en términos de ámbito o alcance. Primero hay que saber que los comentarios están fuera del programa, no están dentro de su ámbito, el compilador no los interpreta, son señales que el programador usa para facilitar la comprensión del código. El ámbito más externo es donde importamos los recursos que se requerirán para el programa y donde declaramos el programa mismo. Aquí está el principio de todo. En Javasiempre desarrollaremos clases, y siempre usaremos clases ya hechas que importaremos. Los recursos mínimos para programar están en el paquete *lang* de la librería *java* que el compilador importa por omisión, aunque aquí (ya lo decíamos en un comentario de la clase*HolaMundoOO*) lo haremos explícitamente por razones didácticas.   
Así que al comenzar a desarrollar un programa debemos primero determinar las clases externas necesarias para ayudar a la clase que crearemos nosotros, y después crear ésta. Para la primera labor usamos la palabra reservada *import* y todas las clases invocadas así podrán ser utilizadas en cualquier lugar del bloque de código de la clase; para la segunda, empleamos las palabras reservadas *public class* seguidas del nombre que deseemos asignarle; debe ser único, preferentemente un sustantivo, iniciar con mayúscula y expresar claramente su función porque será el identificador de la clase; después van dos llaves que contendrán el bloque de código, como muestra la figura siguiente.

  
El rectángulo representa el ámbito o alcance. Los objetos del paquete lang están disponibles en todos lados.   
Dentro del bloque de la clase está el lugar donde deben crearse las propiedades o atributos y declararse los métodos. Las propiedades deben escribirse primero, fuera de cualquier método y su alcance será toda la clase, que es lo que significa el rectángulo interno en la siguiente figura.

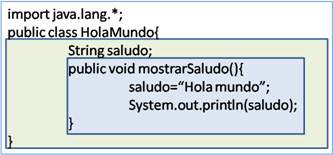
  
Aquí hace falta hablar de la sintaxis de la creación e inicialización de las variables. En Java, toda variable se crea estableciendo su tipo seguido de un nombre, que deberá ser único en el ámbito donde se le declara; se inicializan siempre con una expresión. *Crear* significa asignarles un espacio en memoria, mientas que *inicializar* es darles un valor:

String saludo; → Crea una variable llamada saludo.  
saludo= “Hola mundo”; → La inicializa

Cuando las variables son atributos deben crearse únicamente y debe hacerse fuera de los métodos; por otro lado, serán inicializadas dentro de alguno de ellos en el que convenga. Es posible hacer ambas labores en una sola línea pero será sólo aquéllas que sirven a las tareas de los métodos:

String muestra= “Esto es una muestra”;

Los métodos, por su parte, se declaran estableciendo primero el nivel de acceso. En este caso *mostrarSaludo()* tiene acceso público (para eso usamos la palabra reservada *public*), que significa que cualquier objeto externo puede invocar la tarea encapsulada en el método. Hay otras posibilidades, por ejemplo *prívate*, que es también una palabra reservada, y que significa que el método sólo puede ser usado al interior de la clase en la que es declarado. Después del nivel de acceso, está el tipo de datos que el método devuelve; *mostrarSaludo()*no regresa ningún valor; para indicar esto se usa otra palabra reservada: *void*, que, como veremos después, podrá ser sustituida por cualquier tipo de datos. Todos los métodos deben tener su propio bloque de código, en el que, como ya dijimos, está encapsulada su tarea. El método en cuestión inicializa *saludo*, e invoca la clase *System* (que importamos del paquete *lang* al principio) cuyo atributo *out* es un objeto que tiene el método*println(cadena)* cuya tarea es imprimir una línea con el texto que recibe como parámetro en la salida estándar del sistema. El método *mostrarSaludo()* y su alcance están representados en el rectángulo más interno de la siguiente figura.

  
En cuanto a la sintaxis, falta decir que otra de las responsabilidades de cualquier clase es autoconstruirse por lo que tendrá un método que se llamará igual que la clase misma, deberá ser público, no tiene modificador del tipo que devuelve y lo llamamos el constructor, con lo anterior la clase queda como sigue (obsérvese que en este método se inicializa el atributo):

import java.lang.\*;

public class HolaMundoOO{

String saludo; //La clase Sring la importamos de java.lang

//Creamos el método constructor con el mismo nombre de la clase

public HolaMundoOO(){

saludo="Hola mundo";//En el constructor se inicializan las propiedades

}

public void mostrarSaludo(){

System.out.println(saludo);//La clase System la importamos de java.lang

}

}

Finalmente, otro mecanismo de encapsulamiento es crear paquetes para guardar en ellos las clases que hacen labores afines. Esto se logra con la palabra reservada *package* seguida del nombre que identificará al paquete; éstos se escriben sólo con minúsculas. La versión final de la clase *HolaMundoOO* que queda encapsulada en el paquete *holamundo*, que importa de *java.lang* las clases *String* y *System*, que tiene el atributo *saludo* de tipo *String*, que se autoconstruye e inicializa la propiedad, que tiene un método que muestra saludo usando la clase *System* queda así (Atiéndanse los comentarios):

package holamundo;//Los paquetes son subdirectorios

import java.lang.String;

import java.lang.System;

public class HolaMundoOO{

String saludo;

//Creamos el método constructor con el mismo nombre de la clase

public HolaMundoOO(){

saludo="Hola mundo";//el constructor inicializa las propiedades

}

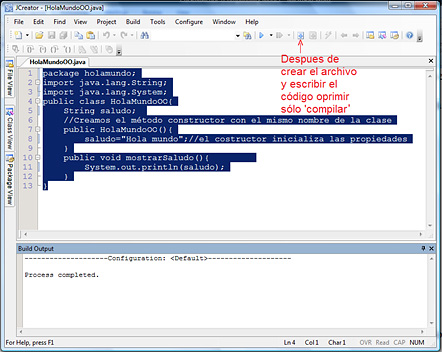
public void mostrarSaludo(){

System.out.println(saludo);

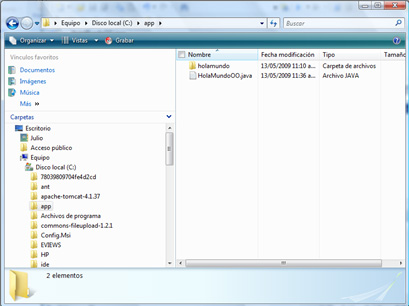
}

}

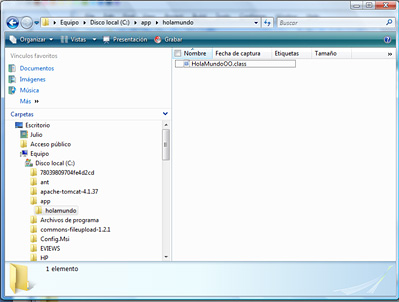
Si aplicamos al programa anterior los mismos pasos del apartado de instalación para probar JCreator, excepto el de ejecución; es decir, crear un nuevo documento vacío con el mismo nombre de la clase, copiar el código anterior y oprimir el botón de compilación debemos obtener algo parecido a esto:



El efecto de compilar es que se crea un archivo .java y el paquete es un subdirectorio:



Dentro del paquete queda el archivo .class, que es el que la máquina virtual puede interpretar:



# 7.1. Propiedades básicas de formularios y elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado forms y que contiene la referencia a todos los formularios de la página.

Para acceder al array forms, se utiliza el objeto document, por lo que document.forms es el array que contiene todos los formularios de la página. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

document.forms[0];

Además del array de formularios, el navegador crea automáticamente un array llamado elements por cada uno de los formularios de la página. Cada array elements contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

document.forms[0].elements[0];

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

document.forms[0].elements[document.forms[0].elements.length-1];

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que "el primer formulario de la página" ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array document.forms.

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (atributo name) o a través de su atributo id. El objeto document permite acceder directamente a cualquier formulario mediante su atributo name:

var formularioPrincipal = document.formulario;

var formularioSecundario = document.otro\_formulario;

<form name="formulario" >

...

</form>

<form name="otro\_formulario" >

...

</form>

Accediendo de esta forma a los formularios de la página, el script funciona correctamente aunque se reordenen los formularios o se añadan nuevos formularios a la página. Los elementos de los formularios también se pueden acceder directamente mediante su atributo name:

var formularioPrincipal = document.formulario;

var primerElemento = document.formulario.elemento;

<form name="formulario">

<input type="text" name="elemento" />

</form>

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos. El siguiente ejemplo utiliza la habitual funcióndocument.getElementById() para acceder de forma directa a un formulario y a uno de sus elementos:

var formularioPrincipal = document.getElementById("formulario");

var primerElemento = document.getElementById("elemento");

<form name="formulario" id="formulario" >

<input type="text" name="elemento" id="elemento" />

</form>

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

* type: indica el tipo de elemento que se trata. Para los elementos de tipo <input> (text, button,checkbox, etc.) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple. Por último, en los elementos de tipo <textarea>, el valor de type es textarea.
* form: es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar document.getElementById("id\_del\_elemento").form
* name: obtiene el valor del atributo name de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
* value: permite leer y modificar el valor del atributo value de XHTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos checkbox y radiobutton no es muy útil, como se verá más adelante

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

* onclick: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (<input type="button">, <input type="submit">, <input type="image">).
* onchange: evento que se produce cuando el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>). También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, lo que técnicamente se conoce como que "el otro campo de formulario ha perdido el foco".
* onfocus: evento que se produce cuando el usuario selecciona un elemento del formulario.
* onblur: evento complementario de onfocus, ya que se produce cuando el usuario hadeseleccionado un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior "ha perdido el foco".

Forma de insertar JavaScript en un Archivo HTML

<script type="text/javascript">   
  
alert("DOM con javascript blabla");   
  
</script>   
  
otra es con un manejador de evento si yo como usuario doy click aquí que haga esto o paso el raton o etc.   
  
  
<input type='button' onclick="var a = 'Hola Mundo'; alert(a)" value='Hola' />   
  
no es ideal pero efectivo si   
  
bueno otra es importar tu archivo en tu html es lo mejor para proyectos grandes te dire que es bueno si la mayoria de librerias usan esto importas y usas a todo archivo que se use la misma operacion   
  
<script type='text/javascript' src='UBICACION DEL ARCHIVO'></script>

La etiqueta es la siguiente:   
  
<script type="text/javascript">   
---- TU CODIGO JAVASCRIPT AQUI -------   
<script/>

Dom y Relacion entre JavaScript y HTML

¿Qué es el Modelo de Objetos del Documento?

*Editores*

Jonathan Robie, Texcel Research

Introducción

El Modelo de Objetos del Documento (DOM) es una interfaz de programación de aplicaciones (API) para documentos HTML y XML. Define la estructura lógica de los documentos y el modo en que se accede y manipula un documento. En la especificación del DOM, el término "documento" se utiliza en un sentido amplio. XML se utiliza cada vez más como un medio para representar muchas clases diferentes de información que puede ser almacenada en sistemas diversos, y mucha de esta información se vería, en términos tradicionales, más como datos que como documentos. Sin embargo, XML presenta estos datos como documentos, y se puede usar el DOM para manipular estos datos.

Con el Modelo de Objetos del Documento los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido. Se puede acceder a cualquier cosa que se encuentre en un documento HTML o XML, y se puede modificar, eliminar o añadir usando el Modelo de Objetos del Documento, salvo algunas excepciones. En particular, aún no se han especificado las interfaces DOM para los subconjuntos internos y externos de XML.

Siendo una especificación del W3C, uno de los objetivos importantes del Modelo de Objetos del Documento es proporcionar un interfaz estándar de programación que pueda utilizarse en una amplia variedad de entornos y aplicaciones. El DOM se ha diseñado para ser utilizado en cualquier lenguaje de programación. Para proporcionar una especificación de las interfaces DOM precisa e independiente del lenguaje, hemos decidido definir las especificaciones en OMG IDL, según se define en la [especificación CORBA 2.2.](http://www.omg.org/corba/corbaiiop.htm) Además de la especificación OMG IDL, proporcionamos correspondencias con los lenguajes Java y ECMAScript (un lenguaje de scripts industrial basado en JavaScript y JScript). *Nota: OMG IDL se usa únicamente como un medio de especificar las interfaces independiente de la plataforma y del lenguaje. Se podrían haber utilizado otros IDLs. En general, los IDLs se diseñan para entornos de computación específicos. El Modelo de Objetos del Documento puede implementarse en cualquier entorno de computación, y no requiere las librerías de enlazado de objetos (object binding runtimes) generalmente asociadas con tales IDLs.* 

Lo que el Modelo de Objetos del Documento es

El DOM es un API de programación para documentos. Guarda una gran similitud con la estructura del documento al que modeliza. Por ejemplo, considérese esta tabla, tomada de un documento HTML:

<TABLE>

<TBODY>

<TR>

<TD>Shady Grove</TD>

<TD>Aeolian</TD>

</TR>

<TR>

<TD>Over the River, Charlie</TD>

<TD>Dorian</TD>

</TR>

</TBODY>

</TABLE>

El DOM representa esta tabla de este modo:

Representación del DOM de la tabla del ejemplo

En el DOM, los documentos tienen una estructura lógica que es muy parecida a un árbol. Para ser más precisos, es más bien como un "bosque" o una "arboleda", que puede contener más de un árbol. Sin embargo, el DOM no especifica que los documentos deban ser *implementados* como un árbol o un bosque, ni tampoco especifica cómo deben implementarse las relaciones entre objetos. El DOM es un modelo lógico que puede implementarse de cualquier manera que sea conveniente. En esta especificación, usamos el término *modelo de estructura* para describir la representación en forma de árbol de un documento, evitando la utilización de términos tales como "árbol" o "bosque" para evitar la implicación de una implementación en particular. Una propiedad importante de los modelos de estructura del DOM es su *isomorfismo estructural*: si dos implementaciones cualesquiera del Modelo de Objetos del Documento se usan para crear una representación del mismo documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

Se eligió el nombre "Modelo de Objetos del Documento" porque es un "modelo de objetos" en el sentido tradicional del diseño orientado a objetos: los documentos se modelizan usando objetos, y el modelo comprende no solamente la estructura de un documento, sino también el comportamiento de un documento y de los objetos de los cuales se compone. En otras palabras, los nodos del diagrama anterior no representan una estructura de datos, sino que representan objetos, los cuales pueden tener funciones e identidad. Como modelo de objetos, el DOM identifica:

* las interfaces y objetos usados para representar y manipular un documento
* la semántica de estas interfaces y objetos, incluyendo comportamiento y atributos
* las relaciones y colaboraciones entre estas interfaces y objetos

Tradicionalmente, la estructura de los documentos SGML se ha representado mediante un modelo de datos abstractos, no con un modelo de objetos. En un modelo de datos abstractos, el modelo se centra en los datos. En los lenguajes de programación orientados a objetos, los datos se encapsulan en objetos que ocultan los datos, protegiéndolos de su manipulación directa desde el exterior. Las funciones asociadas con estos objetos determinan cómo pueden manipularse los objetos, y son parte del modelo de objetos.

El Modelo de Objetos del Documento consiste actualmente de dos partes, el Núcleo del DOM y el DOM HTML. El Núcleo del DOM representa la funcionalidad usada para los documentos XML, y también sirve de base para el DOM HTML. Una implementación conforme del DOM debe implementar todas las interfaces fundamentales del capítulo sobre el Núcleo con la semántica definida. Además, debe implementar o bien el DOM HTML o bien las interfaces extendidas (XML), o ambas, con la semántica definida. 

Lo que el Modelo de Objetos del Documento no es

Se ha incluido esta sección para dar una noción más precisa del DOM distinguiéndolo de otros sistemas que aparentemente pueden resultar similares a él.

* Aunque el Modelo de Objetos del Documento ha recibido una gran influencia del "HTML Dinámico", en el Nivel 1 no implementa todo el "HTML Dinámico". En particular, aún no se han definido los eventos. El Nivel 1 se ha diseñado para establecer un fundamento sólido para esta clase de funcionalidad, por medio de un modelo del propio documento a la vez robusto y flexible.
* El Modelo de Objetos del Documento no es una especificación binaria. Los programas DOM escritos en el mismo lenguaje serán compatibles entre plataformas a nivel de código fuente, pero el DOM no define ninguna forma de interoperabilidad binaria.
* El Modelo de Objetos del Documento no es una manera de ofrecer objetos persistentes para XML o HTML. En lugar de especificar cómo pueden representarse objetos en XML, el DOM especifica cómo se representan los documentos XML y HTML como objetos, de modo que puedan ser utilizados por programas orientados a objetos.
* El Modelo de Objetos del Documento no es un conjunto de estructuras de datos, es un modelo de objetos que especifica interfaces. Aunque este documento contiene diagramas que muestran relaciones padre/hijo, éstas son relaciones lógicas definidas por las interfaces de programación, no representaciones de ninguna estructura interna de datos particular.
* El Modelo de Objetos del Documento no define "la semántica interna real" del XML o del HTML. La semántica de estos lenguajes está definida por las Recomendaciones correspondientes del W3C. El DOM es un modelo de programación diseñado para respetar estas semánticas. El DOM no tiene ninguna consecuencia en el modo en que se escriben los documentos XML y HTML; cualquier documento que pueda escribirse con estos lenguajes puede ser representado en el DOM.
* El Modelo de Objetos del Documento, a pesar de su nombre, no es un competidor del modelo de objetos COM. COM, al igual que CORBA, es una manera independiente del lenguaje de especificar interfaces y objetos; el DOM es un conjunto de interfaces y objetos diseñado para manipular documentos HTML y XML. El DOM se puede implementar usando sistemas independientes del lenguaje como COM o CORBA; también se puede implementar usando enlaces específicos del lenguaje, como los especificados en este documento para Java y ECMAScript.

De dónde vino el Modelo de Objetos del Documento

El DOM se originó como una especificación para permitir que los programas Java y los scripts de JavaScript fueran portables entre los navegadores web. El "HTML Dinámico" fue el ascendiente inmediato del Modelo de Objetos del Documento, y originalmente se pensaba en él principalmente en términos de navegadores. Sin embargo, cuando se formó el Grupo de Trabajo DOM en el W3C, también se unieron a él compañías de otros ámbitos, incluyendo los de la edición y archivo de documentos HTML y XML. Varias de estas compañías habían trabajado con SGML antes de que se hubiera desarrollado el XML; como resultado de ello, el DOM ha recibido influencias de los "Groves" de SGML y del estándar HyTime. Algunas de estas compañías también habían desarrollado sus propios modelos de objetos para documentos a fin de proporcionar un API para los editores o los archivos de documentos SGML/XML, y estos modelos de objetos también han influido en el DOM. 

Las entidades y el núcleo del DOM

En las interfaces fundamentales del DOM no hay objetos que representen entidades. Las referencias numéricas de caracteres y las referencias a entidades predefinidas en HTML y en XML son reemplazadas por el carácter individual que constituye la sustitución de la entidad. Por ejemplo, en:

<p>This is a dog &amp; a cat</p>

el "&amp;" será reemplazado por el carácter "&", y el texto del elemento P formará una única secuencia continua de caracteres. Debido a que las referencias numéricas de caracteres y las entidades predefinidas no son reconocidas como tales en las secciones CDATA, ni en los elementos SCRIPT y STYLE de HTML, no son reemplazadas por el carácter individual al que aparantemente se refieren. Si el ejemplo anterior estuviera contenido en una sección CDATA, el "&amp;" no sería reemplazado por "&"; y el <p> tampoco sería reconocido como una etiqueta inicial. La representación de entidades generales, tanto internas como externas, está definida dentro de las interfaces extendidas (XML) del Nivel 1 de la especificación.

Nota: Cuando la representación DOM de un documento sea serializada como texto XML o HTML, las aplicaciones necesitarán comprobar cada carácter de los datos de texto para ver si necesita ser convertido en una secuencia de escape usando una entidad numérica o predefinida. De lo contrario se podría obtener HTML o XML inválido. Además, las implementaciones deberían ser conscientes del hecho de que la serialización en una codificación de caracteres ("charset") que no cubra completamente la ISO 10646 puede fallar si hay caracteres en el código o en las secciones CDATA que no estén presentes en esa codificación.

Interfaces DOM e Implementaciones DOM

El DOM especifica interfaces que pueden utilizarse para manipular documentos XML o HTML. Es importante darse cuenta de que estas interfaces son una abstracción. Comparables a las "clases de base abstractas" de C++, constituyen un medio de especificar una forma de acceder y manipular la representación interna que una aplicación hace de un documento. Las interfaces no implican una implementación concreta en particular. Cada aplicación DOM es libre de mantener los documentos según una representación conveniente cualquiera, siempre y cuando soporte las interfaces mostradas en esta especificación. Algunas implementaciones del DOM serán programas existentes que usen las interfaces del DOM para acceder a programas escritos mucho antes de que existiera la especificación del DOM. Por tanto, el DOM se ha diseñado para evitar dependencias de la implementación. En particular,

1. Los atributos definidos en el IDL no implican objetos concretos que deban tener miembros de datos específicos - en las correspondencias con cada lenguaje, se transforman en pares de funciones get()/set(), no en un miembro de datos. (Los atributos de sólo lectura sólo tendrán una función get().)
2. Las aplicaciones DOM pueden proporcionar interfaces adicionales y objetos que no se encuentren en esta especificación y seguir siendo considerardas como conformes con el DOM.
3. Debido a que especificamos interfaces, y no los objetos reales que deben ser creados, el DOM no puede saber a qué constructores llamará una implementación. En general, los usuarios del DOM llamarán a métodos createXXX() de la clase Document para crear estructuras del documento, y las implementaciones del DOM crearán sus propias representaciones internas de dichas estructuras en sus implementaciones de las funciones createXXX().

La creación del *Document Object Model* o DOM es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

El Modelo de Objetos del Documento (DOM) permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos que un programa Javascript puede actuar sobre ellos.

### ¿Qué es el Modelo de Objetos del Documento?

Acorde al [W3C](http://www.w3c.es/) el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API) para documentos validos [HTML](http://www.maestrosdelweb.com/editorial/nivelhtml/) y bien construidos [XML](http://www.maestrosdelweb.com/editorial/xmlintro/). Define la estructura lógica de los documentos y el modo en que se accede y manipula.

El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente véase el siguiente ejemplo:

<body>

<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio. </p>

<ul>

<li>Primer punto en la lista</li>

<li>Otro punto en la lista</li>

</ul>

</body>

Como puede verse un elemento [a] se encuentra localizado dentro de un elemento [p] del HTML, convirtiéndose en un nodo hijo, o simplemente hijo del nodo [p], de manera similar [p] es el nodo padre. Los dos nodos li son hijos del mismo padre, llamándose nodos hermanos o simplemente hermanos.

Es importante comprender la diferencia entre elementos y nodos de textos. Los elementos comúnmente son asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como <p>, <img> y <div> por lo que tienen atributos y contienes nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.

### Siempre use el DOCTYPE correcto

El DOCTYPE (abreviado del inglés “document type declaration”, declaración del tipo de documento) informa cual versión de (X)HTML se usará para validar el documento; existen varios tipos a seleccionar. El DOCTYPE, debe aparecer siempre en la parte superior de cada página HTML y siendo un componente clave de las páginas web “obedientes” a los[estándares](http://www.maestrosdelweb.com/editorial/estandaresweb/).

En caso de usarse un DOCTYPE incompleto, no actualizado o simplemente no usarlo llevará al navegador a entrar en modo raro o extraño, donde el navegador asume que se ha programado fuera de los estándares.

Todavía todos los [navegadores](http://www.maestrosdelweb.com/editorial/firefox/) actuales no son capaces de procesar correctamente todos los tipos de documentos, sin embargo, muchos de ellos funcionan correctamente en los navegadores más utilizados actualmente, tales como:

HTML 4.01 Strict y Transitional, XHML 1.0 Strict y Transitional los se comportan del modo correcto en [Internet Explorer](http://www.maestrosdelweb.com/actualidad/3241/) (versión 6, 7 Beta), [Mozilla](http://www.maestrosdelweb.com/editorial/firefox/)y [Opera 7](http://www.maestrosdelweb.com/editorial/opera/). De ahora en adelante se adoptará para cada ejemplo HTML 4.01 Strict :

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Resultando una única línea de código, o dos líneas con un salto de línea después de EN”.

### La importancia de validar el HTML

Si los elementos son anidados de manera inadecuada pueden generarse problemas, véase la siguiente línea:

<p>Estos elementos han sido <strong>incorrectamente </p>anidados </strong>

El árbol que resulta de esto se encuentra incorrectamente anidado del todo, por tanto generará errores inesperados en los navegadores. Manteniendo su HTML válido se pueden evitar tales problemas.

### Accediendo a los elementos

Afortunadamente, Javascript permite acceder a cada uno de los elementos de una página utilizando tan sólo algunos métodos y propiedades.

Si desea encontrar de manera rápida y fácil un elemento se tiene a la mano el método getElementById. El mismo permite un acceso inmediato a cualquier elemento tan sólo conociendo el valor de su atributo id. Véase el siguiente ejemplo:

<p>

<a id="contacto" href="contactos.html">Contáctenos</a>

</p>

Puede usarse el atributo id del elemento a para acceder al mismo:

var elementoContacto = document.getElementById("contacto");

Ahora el valor de la variable elementoContacto está referida al elemento [a] y cualquier operación sobre la misma afectará el hiperenlace.

El método getElementById es adecuado para operar sobre un elemento en específico, sin embargo, en ocasiones se necesita trabajar sobre un grupo de elementos por lo que en este caso puede utilizarse el método getElementsByTagName. Este retorna todos los elementos de un mismo tipo. Asumiendo la siguiente lista desordenada:

<ul>

<li><a href="editorial.html">Editorial</a></li>

<li><a href="semblanza.html">Autores</a></li>

<li><a href="noticias.html">Noticias</a></li>

<li><a href="contactos.html">Contátenos</a></li>

</ul>

Puede obtenerse todos los hipervínculos de la siguiente manera:

var hipervinculos= document.getElementsByTagName("a");

El valor de la variable hipervinculos es una colección de elementos [a]. Las colecciones son arreglos pudiéndose acceder a cada elemento a través de la ya conocida notación con corchetes.

Los elementos devueltos por getElementsByTagName serán ordenado según el orden que aparezcan en el código fuente. Por tanto para el caso anterior quedaría así:

* hipervinculos[0] el elemento [a] para “Editorial”
* hipervinculos[1] el elemento [a] para “Autores”
* hipervinculos[2] el elemento [a] para “Noticias”
* hipervinculos[3] el elemento [a] para “Contáctenos”

Otra maneras de acceder a un elemento usando su id es document.all["id"] la cual fue introducida en Internet Explorer 4 y document.layers["id"] introducida por Netscape 5 por que el W3C todavía no había estandarizado la manera de acceder a los elementos mediante su id. Sin embargo, no se recomienda su uso porque al estar fuera de los estándares actuales hay navegadores que no soportan estos métodos.

Por otro lado existen varios elementos en un documento HTML que pueden ser accedidos de otras maneras. El elemento body de un documento puede accederse a través de la forma document.body, mientras que el conjunto de todos los formularios en un documento puede encontrase en document.forms, así mismo el conjunto de todas las imágenes sería mediante document.images.

Actualmente la mayoría de los navegadores soportan esto métodos aún así es recomendable el uso del método getElementsByTagName, véase el siguiente ejemplo para acceder al elemento body:

var body = document.getElementsByTagName("body")[0];

### Creando elementos y textos

La creación de nodos es posible mediante el uso de dos métodos disponibles en el objeto document. Dichos métodos son:

* createElement(Tipo cadena): Crea un nuevo elemento del tipo especificado y devuelve un referencia a dicho elemento.
* createTextNode(Cadena de texto): Crea un nuevo nodo de texto con el contenido especificado en la cadena de texto.

El siguiente ejemplo muestra cómo se crea un nuevo elemento de párrafo vacío:

var nuevoEnlace = document.createElement("a");

La variable nuevoEnlace ahora referencia un nuevo enlace listo para ser insertado en el documento. El texto que va dentro del elemento [a] es un nodo de texto hijo, por lo que debe ser creado por separado.

var nodoTexto = document.createTextNode("Semblanza");

Luego si desea modificar el nodo de texto ya existente, puede utilizarse la propiedad nodeValue, esta permite coger y poner el nodo de texto:

var textoViejo = nodoTexto.nodeValue;

nodoTexto.nodeValue = "Novedades";

El valor de la variable textoViejo es ahora "Semblanza" y el nuevo texto "Novedades". Se puede insertar un elemento o texto (nodo) como último hijo de un nodo ya existente usando el método appendChild. Este método coloca el nuevo nodo después de todos los hijos del nodo.

NuevoEnlace.appendChild(nodoTexto);

Ahora todo lo que se necesita es insertar el enlace en el cuerpo del documento. Para hacer esto, se necesita una referencia al elemento body del documento, teniendo como guía los estándares siguientes:

var cuerpoRef = document.getElementsByTagName("body")[0];

cuerpoRef.appendChild(nuevoEnlace);

Otra manera sería utilizando el método getElementById. Para ello se asume que la etiqueta <body> tiene asignado un valor para el atributo id.

<body id=”cuerpo”>

var cuerpoRef = document.getElementById("cuerpo");

cuerpoRef.appendChild(nuevoEnlace);

Existen básicamente tres maneras mediante las cuales un nuevo elemento o nodo de texto puede ser insertado en una página Web. Todo ello depende del punto en el cual se desee insertar el nuevo nodo: como último hijo de un elemento, antes de otro nodo o reemplazo para un nodo.

El caso de apertura de un nuevo hijo ya fue visto en el ejemplo anterior, luego para insertar el nodo antes de otro nodo se realiza utilizando el método insertBefore de su elemento padre, mientras que el reemplazo de nodo se utiliza el método replaceChild de su elemento padre.

Al usar insertBefore, se necesita tener referencias al nodo que va ser insertado y donde va a ser insertado, considérese el siguiente código HTML:

<p id="mwEnlaces">

<a id="editor" href="editorial.html">Editorial</a>

</p>

Luego el nuevo enlace será insertado antes de enlace ya existente llamando el método insertBefore desde el nodo padre (párrafo):

var anclaTexto = document.createTextNode("Actualidad");

var nuevoAncla = document.createElement("a");

nuevoAncla.appendChild(anclaTexto);

var anclaExistente = document.getElementById("editor");

var padre = anclaExistente.parentNode;

var nuevoHijo = padre.insertBefore(nuevoAncla, anclaExistente);

Si se hiciera una traducción del DOM hacia HTML después de esta operación el resultado sería el siguiente:

<p id="mwEnlaces">

<a> Actualidad </a><a id="editor" href="editorial.html">Editorial</a>

</p>

En el caso de reemplazar el enlace usando replaceChild:

var nuevoHijo = padre.replaceChild(nuevoAncla, anclaExistente);

El DOM lucirá así:

<p id="mwEnlaces">

<a> Actualidad </a>

</p>

### Usando innerHTML

En aplicaciones complejas donde es necesario crear varios elementos a la vez, el código JavaScript generado puede ser extenso recurriéndose a la propiedad innerHTML. Dicha propiedad fue introducida por [Microsoft](http://www.maestrosdelweb.com/editorial/microsoft/) permitiendo leer y escribir el contenido HTML de un elemento.

Por ejemplo, puede crearse fácilmente una [tabla](http://www.maestrosdelweb.com/editorial/tutsql11/) con múltiples celdas e insertarla luego en la página con innerHTML:

var tabla = '<table border="0">';

tabla += '<tr><td>Celda 1</td><td>Celda 2</td><td> Celda 3</td></tr>';

tabla += '</table>';

document.getElementById("datos").innerHTML = tabla;

### Eliminando un elemento o nodo de texto

Se pueden eliminar nodos existentes y nuevos. El método removeChild permite eliminar nodos hijos a cualquier nodo con tan sólo pasarle las referencias del nodo hijo [a] eliminar y su correspondiente padre. Para mejor compresión retómese el ejemplo anterior:

<p id="mwEnlaces">

<a id="editor" href="editorial.html">Editorial</a>

</p>

El método removeChild será usado para eliminar el hipervínculo del elemento padre párrafo:

var ancla = document.getElementById("editor");

var padre = ancla.parentNode;

var hijoRemovido = padre.removeChild(ancla);

La variable hijoRemovido todavía hace referencia al elemento, de manera que fue removido pero no destruido, no pudiéndose localizar en ninguna parte del DOM. Este se encuentra disponible en memoria como si fuera creado usando el método createElement. Esto permite posicionarlo en cualquier otra parte de la página.

### Lectura y escritura de los atributos de un elemento

Las partes más frecuentemente usadas de un elemento HTML son sus atributos, tales como: id, class, href., title, [estilos CSS](http://www.maestrosdelweb.com/editorial/cssplan/), entre muchas otras piezas de información que pueden se incluidas en una etiqueta HTML.

Los atributos de una etiqueta son traducidos por el navegador en propiedades de un objeto. Dos métodos existen para leer y escribir los atributos de un elemento, getAttribute permite leer el valor de un atributo mientras que setAttribute permite su escritura.

En ocasiones se hace necesario ver las propiedades y métodos de un determinado elemento, esto puede realizarse mediante la siguiente función utilitaria:

function inspector(el) {

var str =””;

for (var i in el){

str+=I + “: ” + el.getAttribute(i) + “\n”;

}

alert(str);

}

Para usar la función inspector() tan sólo debe pasarle la referencia al elemento, continuando con el ejemplo anterior resulta:

var ancla = document.getElementById("editor");

inspector(ancla);

Para modificar el atributo title del hipervínculo, elemento referenciado por la variable ancla, se usará el setAttribute, pasándole el nombre del atributo y el valor:

var ancla = document.getElementById("editor");

ancla.setAttribute("title", "Artículos de programación");

var nuevoTitulo = ancla.getAttribute("title");

El valor de la variable nuevoTitulo es ahora “Artículos de programación”.

### Manipulando los estilos de los elementos

Como se ha visto, los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. Las propiedades de estilo pueden ser aplicadas a través del DOM.

Cada atributo [CSS](http://www.maestrosdelweb.com/editorial/introcss/) posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS pero sin los guiones y llevando la primera letra de las palabras a mayúsculas. Véase el siguiente ejemplo para mayor entendimiento donde se utiliza un atributo CSS modelo:

algun-atributo-css

Tendrá como equivalente la siguiente propiedad o método en Javascript:

algunAtributoCss

Por tanto, para cambiar el atributo CSS font-family de un elemento, podría realizarse de lo siguiente:

ancla.style.fontFamily = 'sans-serif';

Los valores CSS en Javascript serán en su mayoría del tipo cadena; por ejemplo: font-size, pues posee dimensiones tales como “px”, “%”. Sólo los atributos completamente numéricos, tales como z-index serán del tipo entero.

En muchos casos es necesario aparecer y desaparecer un determinado elemento, para ellos se utiliza el atributo CSS display, por ejemplo, para desaparecer:

ancla.style.display = 'none';

Luego para volverlo a mostrar se le asigna otro valor:

ancla.style.display = 'inline';

### Ejemplo: Adjuntar múltiples ficheros a la vez

Este es el primer ejemplo completo donde se propone utilizar la manipulación del DOM mediante javascript con el objetivo de adicionar tantos elementos input del tipo file como tantos ficheros se deseen subir al servidor.

Se muestra una versión simplificada del problema limitada tan sólo al lado del cliente. Para ello es necesario imaginarse un sistema en línea donde se suben ficheros al servidor, ejemplo de ello podría ser una aplicación de [correo electrónico](http://www.maestrosdelweb.com/editorial/emailhis/).

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>Ejemplo para adjuntar múltiples ficheros</title>

<script language="javascript" type="text/javascript">

function nuevoFichero() {

var input = document.getElementsByTagName("input")[0];

var nuevoInput = input.cloneNode(true);

input.parentNode.appendChild(nuevoInput);

}

</script>

</head>

<body>

<form action="upload.php" method="post" enctype="multipart/form-data">

<fieldset><legend>Adjuntar múltiples ficheros</legend>

<input name="ficheros[]" type="file" size="60" >

</fieldset>

<a href="javascript: nuevoFichero();">Adjuntar otro fichero</a>

<input name="Subir" type="submit" value="Adjuntar" >

</form>

</body>

</html>

El enlace para adjuntar otro fichero hace una llamada a la función nuevoFichero() realizándose las siguientes tareas en la misma:

Se accede al primer elemento input encontrado en el documento:

var input = document.getElementsByTagName("input")[0];

* Se crea un nuevo elemento input referenciado por la variable nuevoInput utilizando el método cloneNode resultando un elemento idéntico al primero:

var nuevoInput = input.cloneNode(true);

* Aquí se accede al nodo padre del elemento input mediante el método parentNode y la vez se inserta un nuevo elemento hijo copia del primero por medio del método appendChild:

input.parentNode.appendChild(nuevoInput);

Es de notar que en este ejemplo podría haberse usado el evento clic para la llamada de la función, si embargo se dejó reservado para cuando el tema de la manipulación de eventos del DOM sea abordado.

### Ejemplo: Mostrador de diapositivas

Este ejemplo contempla un mostrador de diapositivas (en este caso imágenes). Se tienen dos enlaces, uno mostrar la diapositiva siguiente y otro para mostrar la diapositiva anterior. Una posible aplicación práctica podría ser en una galería de fotos.

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>Ejemplo de un mostrador de diapositivas</title>

<script type="text/javascript">

var contador = 0;

var diapositivas = [];

function inicio() {

var contenedor = document.getElementById("diapositivas");

while (contenedor.childNodes.length > 0){

if (contenedor.getElementsByTagName("img")[0]==contenedor.firstChild){

diapositivas[diapositivas.length] = contenedor.removeChild(contenedor .firstChild);

}

else contenedor.removeChild(contenedor.firstChild);

}

}

function adelante() {

contador++;

if (contador >= diapositivas.length) contador = 0;

ponerImagen()

}

function atras() {

contador--;

if (contador < 0 ) contador = diapositivas.length - 1;

ponerImagen();

}

function ponerImagen() {

var contenedor = document.getElementById("diapositivas");

if (contenedor.childNodes.length==0)

contenedor.appendChild(diapositivas[contador]);

else contenedor.replaceChild(diapositivas[contador], contenedor.childNodes[0]);

}

</script>

</head>

<body>

<a href="javascript: atras();">Atrás</a>

<span id="diapositivas">

<img src="diapositiva1.jpg" alt="Diapositiva 1" height="100" width="200">

<img src="diapositiva2.jpg" alt="Diapositiva 2" height="100" width="200">

<img src="diapositiva3.jpg" alt="Diapositiva 3" height="100" width="200">

</span>

<a href="javascript: adelante();">Adelante</a>

</body>

</html>

<script type="text/javascript">

inicio();

ponerImagen();

</script>

Este ejemplo se analizará de manera abreviada dejando un análisis detallado por parte del lector. Se tiene dos variables globales, contador y diapositivas, la primera para indicar la diapositiva que se está mostrando actualmente y la última para almacenar en un arreglo el conjunto de diapositivas a mostrar.

La función inicio() remueve todos elementos img y los almacena. Es de notar que todo nodo hijo del nodo con id diapositivas parásito (ejemplo: salto de línea) es eliminado para evitar un mal funcionamiento posterior.

Por otro lado la función ponerImagen() se encarga de colocar la imagen apuntada por el contador en el contenedor de diapositivas. Mientras que las funciones atras() y adelante() se encargan de decrementar e incrementar el contador respectivamente.

### Conclusión

El presente material tan sólo ha sido una ligera aproximación a la manipulación del Modelo de Objetos del Documento. Explorando el DOM se puede encontrar, cambiar, adicionar y eliminar elementos de un documento. Es sin duda alguna una poderosa técnica para generar aplicaciones Web dinámicas en el lado del cliente.