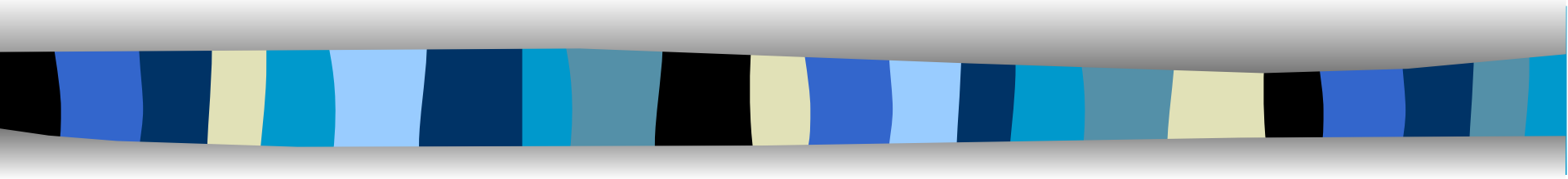
Fundamentos del DOM



JavaScript

Introducción.

- El Modelo de objetos de documento (DOM) ofrece a los programadores un acceso sin precedentes a HTML y les permite manipular y ver HTML como un documento XML.
- El DOM representa la evolución de HTML dinámico, proyecto pionero de Microsoft y Netscape, hacia una verdadera solución entre plataformas independiente del lenguaje.

Jerarquía de nodos

- El **DOM** es una **representación** del documento en forma de **árbol**. De modo que cada elemento estará representado por un nodo. El DOM define la interfaz node y varios tipos de nodos para representar los distintos aspectos del código XML:
  - **Document**: El nodo de nivel superior al que se conectan los demás nodos.
  - **DocumentFragment**: Se puede utilizar como Document para almacenar otros nodos.
  - **Element**: Representa los contenidos de una etiqueta de apertura y otra de cierre, como en <etiqueta></etiqueta>. Este tipo de nodo sólo puede incluir atributos y nodos secundarios.
  - **Attr**: Representa un atributo de par de nombre y valor. No puede incluir nodos secundarios.
  - **Text**:Representa texto sin procesar en un documento XML incluido dentro de etiquetas de apertura y de cierre. Este tipo de nodo no puede tener nodos secundarios.

Definición del DOM

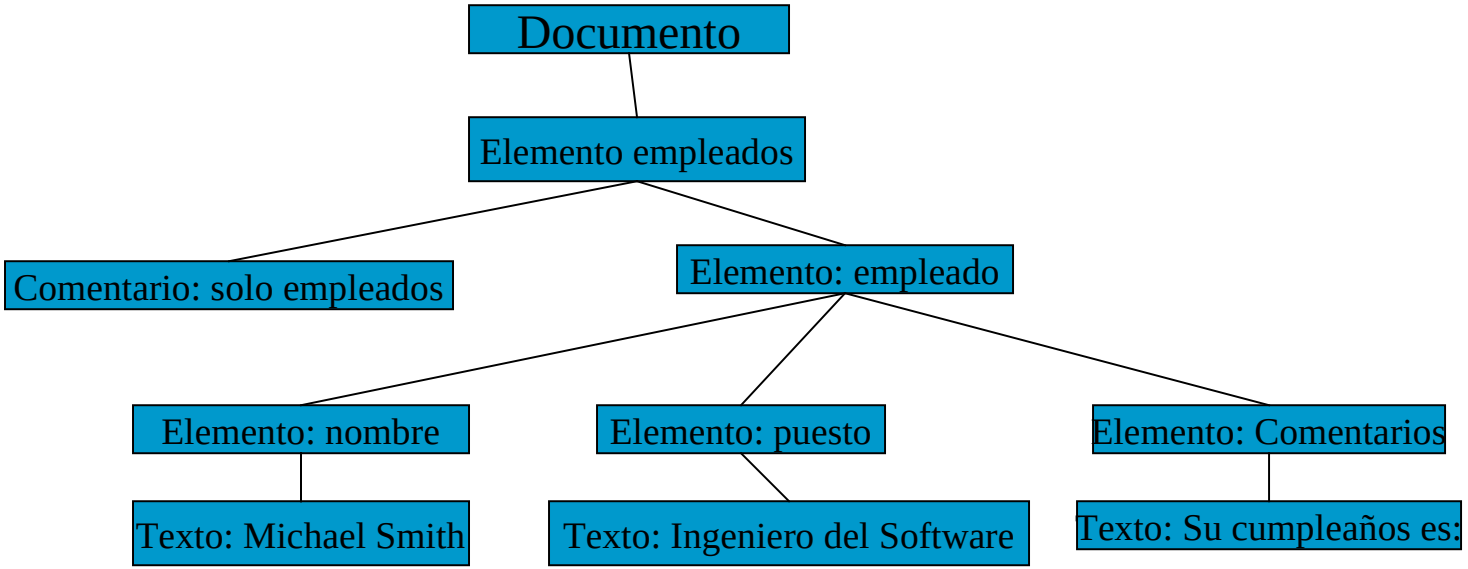
- El Modelo de objetos de documento (DOM) es un API de estructura en árbol esta definido para XML. No sólo se centra en analizar código XML, sino en representar dicho código por medio de una serie de objetos vinculados a los que se puede acceder y se pueden modificar sin tener que volver a analizar el código.

- El DOM es un API independiente del lenguaje, es decir, sin vinculaciones a Java, JavaScript o a otro lenguaje para su implementación. Sin embargo nos centraremos en la implementación de JavaScript.

Jerarquía de nodos

```
<?xml version=“1.0”?>
<empleados>
    <!-- Solo empleados-->
    <empleado>
        <nombre>Michael Smith</nombre>
        <puesto> Ingeniero de Software</puesto>
        <comentarios>Su fecha de nacimientoes: el 8/14/68
        </comentarios>
    </empleado>
</empleados>
```

Jerarquía de nodos. Ejemplo.



Jerarquía de nodos

- La interfaz Node también define un conjunto de propiedades y métodos que todos los tipos de nodo contienen y que se enumeran en la siguiente tabla:

Propiedad/Método	Tipo/Tipo devuelto	Descripción
nodeName	String	El nombre del nodo; se define en función del tipo de nodo.
nodeValue	String	El valor del nodo; se define en función del tipo de nodo
nodeType	Number	Uno de los valores constantes de tipo de nodo.
ownerDocument	Document	Puntero al documento al que pertenece este nodo.
firstChild	Node	Puntero al primer nodo de la lista childNodes.
lastChild	Node	Puntero al último nodo de la lista childNodes

Jerarquía de nodos

Propiedad/Método	Tipo/Tipo devuelto	Descripción
childNodes	NodeList	Una lista de todos los nodos secundarios
previousSibling	Node	Puntero al pariente anterior; null en caso de que sea el primer pariente.
nextSibling	Node	Puntero al siguiente pariente; null en caso de que sea el último pariente.
hasChildNodes()	Boolean	Devuelve true cuando childNodes contiene uno o más nodos.
attributes	NamedNodeMap	Contiene objetos Attr que representan los atributos de un elemento, sólo se utiliza con nodos Element.
appendChild(nodo)	Node	Añade nodo al final de childNodes.
removeChild(nodo)	Node	Elimina nodo de childNodes.
replaceChild(nuevo_nodo, nodo_antiguo)	Node	Sustituye nodo_antiguo en childNodes por nuevo_nodo.
insertBefore(nuevo_nodo,nodo_ref)	Node	Añade nuevo_nodo por delante de nodo_ref en childNodes.

Jerarquía de nodos

- Ademas de los nodos, el DOM también define objetos de ayuda, que se utilizan para trabajar con nodos pero que no forman parte necesariamente de un documento DOM.
  - ***NodeList***: Una matriz de nodos indexada numéricamente; se utiliza para representar los nodos secundarios de un elemento.
  - **NamedNodeMap**: Una matriz de nodos indexados numéricamente y por nombre; se usa para representar atributos de elementos.
- Estos objetos de ayuda proporcionan métodos adicionales de acceso y recorrido para trabajar con documentos DOM. Las aplicaciones concretas de los mismos se analizarán más adelante.

DOM específicos del lenguaje

- Cualquier lenguaje basado en XML, como XHTML, puede utilizar el DOM básico que acabamos de ver porque técnicamente se trata de XML.
- Muchos lenguajes definen sus propios DOM que amplían el núcleo XML para ofrecer características específicas del lenguaje.
- El W3C desarrollo un DOM más específico para XHTML (y HTML). Este DOM define **HTMLDocument** y **HTMLElement** como base de la implementación. Cada elemento HTML se representa por su propio tipo HTMLElement, como por ejemplo **HTMLDivElement** para representar <**div**>.

Acceder a nodos relativos

- En los siguientes apartados utilizaremos esta página HTML:

```
<html>
<head>
<title>Ejemplo de Dom</title>
</head>
<body>
<p>Hola Mundo</p>
<p> Esto es un ejemplo no muy interesante</p>
<p>Aprenderemos a usar el DOM!</p>
</body>
</html>
```

Acceder a nodos relativos

- Para acceder a un elemento <html /> (que es el elemento ***document*** de este archivo), podemos utilizar la propiedad ***documentElement*** de ***document***.

  var oHtml =document.documentElement

  La variable oHtml contiene un objeto HTMLElement que representa a <html/>.

  var oHead = oHtml.firstChild; var oHead = oHtml.childNodes[0];
  var oBody = oHtml.lastChild; var oBody = oHtml.childNodes[1];

Acceder a nodos relativos

- Para obtener el número de nodos secundarios:
  - alert(oHtml.**childNodes**.**length**); //devulve “2”
- El método formal para recuperar nodos secundarios de la lista childNodes es **item():**

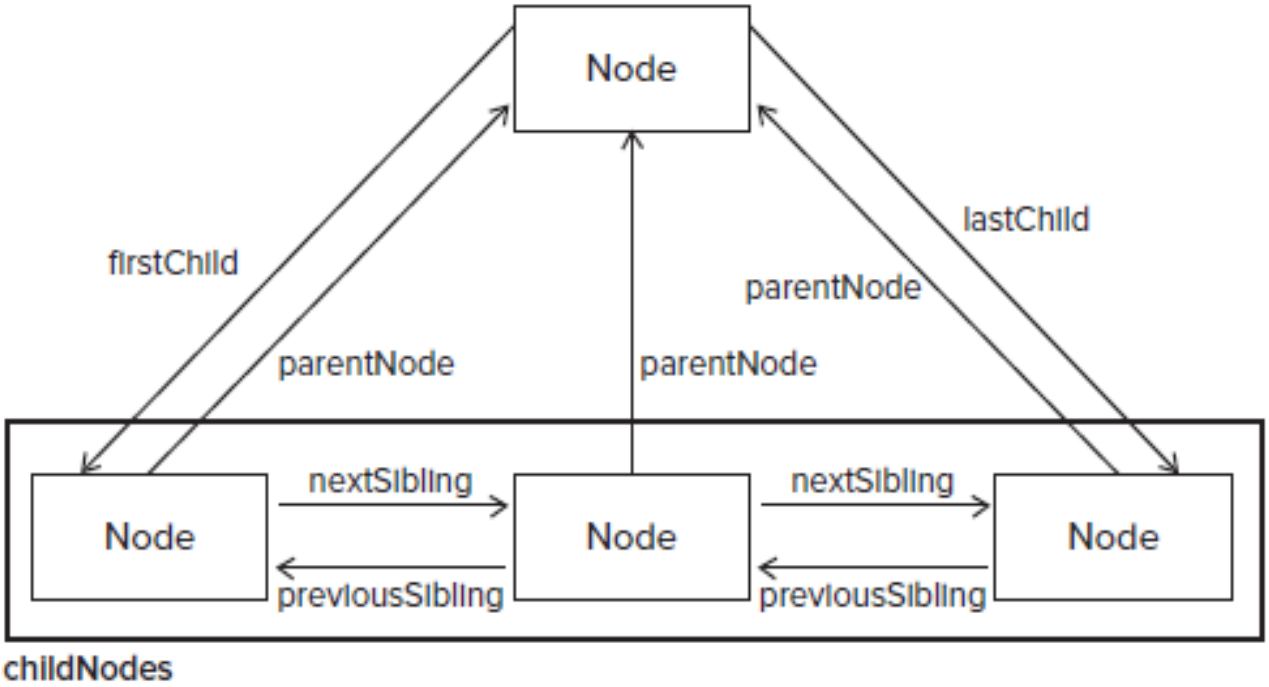
  *var oHead = oHtml.**childNodes.item(0);***

  *var oBody = oHtml.**childNodes.item(1);***

  Con las tres variables oHtml, oHead y oBody podemos experimentar para determinar las relaciones entre las mismas:

  ```
  alert(oHead.parentNode == oHtml);//devuelve true
  alert(oBody.parentNode == oHtml);//devuelve true
  alert(oBody.previusSibling == oHead);//devuelve true
  alert(oHead.nexSibling == oBody);//devuelve true
  alert(oHead.ownerDocument == document);//devuelve true
  ```

Jerarquía de nodos. Relaciones entre nodos.



Acceder a nodos concretos

- Cuando queremos acceder a un nodo concreto y de manera directa podemos utilizar los siguientes métodos:
  - **getElementsByTagName().**
  - **getElementsByName().**
  - **getElementById().**
- Estos métodos nos permiten acceder a cada nodo concreto del documento haciendo referencia a su nombre de etiqueta (tagName), a su nombre (name) o a su identificador (id).

Acceder a nodos concretos. Por su tagName.

- **GetElementsByTagName**: Devuelve un ***NodeList*** de todos los objetos ***Element*** cuya propiedad ***tagName*** sea igual a su valor concreto. En un elemento ***Element.***

  var oImgs = document.getElementsByTagName("img");

- Una vez alamacenadas en oImgs, podemos acceder individualmente a las mismas igual que accedemos a nodos secundarios, es decir, por medio de corchetes o con el método item() (getElementsByTagName() devuelve NodeList, al igual que childNodes):

  alert(oImgs[0].tagName); //devuelve "IMG"

Por alguna razón la mayoría de los navegadores devuelven el nombre de etiqueta en mayúsculas aunque las convenciones de XML dictan lo contrario.

getElementsByTagName. Ejemplo.

- Pretendemos obtener las imágenes del primer párrafo de una pagina. Para ello podría invocar getElementsByTagName() en el primer elemento de párrafo, de esa forma:
  - Var oPs = document.getElementsByTagName(“p”);
  - Var oImgsInp =oPs[0].getElementsByTagName(“img”);
- Podemos utilizar este método para desplazarse hasta cualquier elemento de document o para obtener todos los elementos de document por medio de un asterisco:

  Var oAlElements = document.getElementsByTagName(“*”);

  Este método devuelve todos los elementos incluidos en documento independientemente de sus nombres de etiqueta.

Acceder a nodos concretos. Por su Nombre.

- ***getElmentsByName():*** Permite recuperar todos los elementos en los que el atributo name se establece en un valor concreto.Definido por el DOM HTML. Veamos un ejemplo:

```
<html >
<head>
<title>Ejemplo del DOM</title>
</head>
<body>
   <form method="post" action="haceralgo.php">
     <fieldset>
          <legend> Que color te gusta mas</legend>
          <input  type="radio" name="radColor" value="rojo"/>rojo<br />
          <input  type="radio" name="radColor" value="verde"/>verde<br />
          <input  type="radio" name="radColor" value="azul"/>azul<br />
     </fieldset>
     <input type="submit" value="enviar"/>
   </form>
</body>
</html>
```

Acceder a nodos concretos. Por su Nombre.

- Esta página pregunta al usuario qué color prefiere. Todos los botones de opción tienen el mismo atributo name. Para obtener las referencias a todos los elementos de botón de opción, podríamos utilizar los siguiente:

```
var oRadios = document.getElementsByName("radColor");
```

Tras ello, podemos manipular los botones de opción de la misma forma que si se tratara de otro elemento.

```
alert(oRadios[0].getAttribute("value")); //devuelve "rojo"
```

Acceder a nodos concretos. Por su identificador.

- getElementById(): El segundo método definido por el DOM HTML. Devuelve un elemento con su atributo id establecido en un valor concreto. En XHTML, el atributo id es exclusivo.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>HolA MUNDO</title>
</head>
<body onload="capa()">
    <p>Hola Mundo!</p>
    <div id="div1">Esta es mi primera capa</div>
</body>

</html>
```

Acceder a nodos concretos. Por su identificador.

- La siguiente instrucción nos permitiría acceder al elemento div directamente.

```
var oDiv1 = document.getElementById("div1");
```

Nuevos Selectores.

Para elevar JavaScript al nivel que las circunstancias requieren, se incorporaron nuevas alternativas. Ahora podemos seleccionar elementos HTML aplicando toda clase de selectores CSS por medio de los nuevos métodos **querySelector()** y **querySelectorAll**().

Estos selectores nos permiten seleccionar elementos de manera individual con querySelector() o en una colección con querySelectorAll(). Los dos aceptan como argumento selector CSS.

Nuevos Selectores. querySelector()

- Este método retorna el primer elemento que concuerda con el grupo de selectores especificados entre paréntesis. Los selectores son declarados usando comillas y la misma sintaxis CSS, como en el siguiente ejemplo:

```
function hacerclic(){
document.querySelector("body
p:firstchild").onclick=mostraralerta;
}
function mostraralerta(){
alert('hizo clic!');
}
window.onload=hacerclic;
```

Nuevos Selectores. QuerySelectorAll()

- El método querySelectorAll() devuelve todos los elementos que concuerdan con el grupo de selectores declarados entre paréntesis, en lugar de sólo el primero como hacía el método anterior. El valor devuelto es un array que contiene cada elemento encontrado en el orden en el que aparecen en el documento:

```
function hacerclic(){
var lista=document.querySelectorAll("body p");
lista[0].onclick=mostraralerta;
}
function mostraralerta(){
alert('hizo clic!');
}
window.onload=hacerclic;
```

Crear y manipular Nodos

- Hasta el momento hemos visto como acceder a distintos nodos dentro de un documento, pero sólo es el principio de lo que podemos conseguir con el DOM.

Crear y manipular Nodos .Crear nuevos nodos

- El objeto **Document** de DOM contiene una serie de métodos diseñados para crear distintos tipos de métodos, aunque el objeto document del navegador no los admita todos en todos los navegadores.
- Nos centraremos en los métodos más utilizados que son ***createDocumentFragment***(), ***createElement***() y ***createTextNode***(); los demás no resultan útiles o no cuentan con suficiente compatibilidad para resultar útiles por el momento.

createElement(), createTextNode() y appendChild().

Dado el siguiente código HTML:

```
<html>
   <head>
      <title>ejemplo de
     CreateElement()</title>
   </head>
   <body>
   </body>
</html>
```

A esta página se le desea añadir el siguiente código:

*** Hola mundo ***

1.Creamos el elemento .

var oP = document.createElement("p");

2.Creamos el nodo de texto:

var oText = document.createTextNode("Hola mundo");

3.Ahora debemos conectar el elemento con su elemento texto.

oP.appendChild(oText);

4.Ahora pasamos a conectarlo al elemento document.body o a uno de sus secundarios:

document.body.appendChild(oP);

removeChild(), replaceChild() e insertBefore()

- ***RemoveChild():*** Este método acepta como argumento el nodo a eliminar, y devuelve dicho nodo como valor de la función.
- Si disponemos de una página a la que queramos eliminar un mensaje:

removeChild().

removeChild(), replaceChild() e insertBefore()

- Si deseamos substituir este mensaje por otro nuevo, tendriamos que utilizar el método replaceChild(), que tiene dos argumentos: el nodo que añadir y el nodo que eliminar. replaceChild().
- En el ejemplo, se sustituye el mensaje “Hola Mundo por... otro mensaje.
- Si quisiéramos que el nuevo mensaje apareciera por detrás del anterior, utilizaríamos el método appendChild().
- Si deseáramos que el nuevo mensaje apareciera antes que el antiguo, tendríamos que emplear insertBefore(), método que acepta dos argumentos:
  - El nuevo nodo que añadir y el nodo que debe incluirse por delante.

CreateDocumentFragment()

- Al añadir nodos a document.body (o a uno de sus ancestros), la página se actualiza para reflejar los cambios. Cuando la cantidad de datos a a añadir es grande, el proceso puede ralentizarse si los cambios se realizan de forma individual. Para evitar esta situación, puede crear un fragmento de documento al que añadir los nuevos nodos y, posteriormente, añadir a document el contenido de dicho fragmento. createDocumentFragment()

Características del DOM HTML

- Las propiedades y métodos del DOM básico son genéricas, diseñadas para funcionar con cualquier documento DOM en cualquier situación.
- Las propiedades y métodos del DOM HTML son específicas de HTML y facilitan determinadas operaciones del DOM.
- Por ejemplo: Acceder a ***atributos como propiedades*** además de a propiedades y métodos específicos de elementos para realizar tareas comunes, ***tratar tablas***, de forma más sencilla.

Atributos como propiedades.

- En la mayoría de los casos, todos los atributos de elementos del DOM HTML se incluyen como propiedades:

  ******

- Para obtener y establecer los atributos src y borders con el DOM básico, tendríamos que utilizar los métodos getAttribute() y setAttribute():
- Sin embargo, con el DOM HTML, podemos obtener y establecer estos valores si utilizamos propiedades del mismo nombre:

```
alert(oImg.src);
alert(oImg.border);
oImg.src ="mypicture2.jpg";
oImg.border ="1";
```

Atributos como propiedades.

- El único caso en el que el nombre de atributo no coincide con el nombre de propiedad es con el atributo class, que especifica una clase CSS que aplicar a un elemento, como se indica a continuación.

  *** ***

- Como class es una palabra reservada en ECMAScript, no se puede utilizar como nombre de variable, función o propiedad en JavaScript. Por ello, la propiedad es className:

  alert(oDiv.classname);

  oDiv.className ="footer";

Atributos como propiedades. Propiedades CSS en DOM.

- Las propiedades *CSS* también se pueden acceder de forma directa a través del nodo *DOM*. Sin embargo, en este caso se requiere cierta transformación en el nombre de algunas propiedades.
- En primer lugar, las propiedades *CSS* se acceden a través del atributo style del elemento:

```
var parrafo = document.getElementById("parrafo");
    parrafo.style.margin = "10px";
     <p id="parrafo">...</p>
```

- En el caso de las propiedades *CSS* con nombre compuesto, su nombre se transforma a la notación *camelCase* típica (se eliminan los guiones y la 1ª letra de cada palabra irá en mayúscula salvo la primera palabra).

Métodos de Tablas.

- Imaginemos que queremos crear la siguiente tabla HTML con ayuda del DOM:

```
<html>
<head>
  <title></title>
</head>
<body>

</body>
</html>
```

```
<table width="100%" border="1">
  <tr>
    <td>Celda 1,1</td>
    <td>Celda 1,2</td>
  </tr>
  <tr>
    <td>Celda 2,1</td>
    <td>Celda 2,2</td>
  </tr>
</table>
```

Métodos de Tablas.

- Para realizar esta operación con los métodos del DOM básico, debemos usar el siguiente código

```
//cree a tabla
  var oTable =document.createElement("table");
  oTable.setAttribute("border","1");
  oTable.setAttribute("width","100%");
  //cree el cuerpo
  var oTbody
=document.createElement("tbody");
  oTable.appendChild(oTbody);

  //cree la primera fila
  var oTr1 =document.createElement("tr");
  oTbody.appendChild(oTr1);
  var oTd11 =document.createElement("td");

oTd11.appendChild(document.createTextNode("
celda 1,1"));
  oTr1.appendChild(oTd11);
  var oTd12 =document.createElement("td");

oTd12.appendChild(document.createTextNode("
celda 1,2"));
  oTr1.appendChild(oTd12);
```

```
//cree la segunda fila
  var oTr2 =document.createElement("tr");
  oTbody.appendChild(oTr2);
  var oTd21 =document.createElement("td");

oTd21.appendChild(document.createTextNod
e("celda 2,1"));
  oTr2.appendChild(oTd21);
  var oTd22 =document.createElement("td");

oTd22.appendChild(document.createTextNod
e("celda 2,2"));
  oTr2.appendChild(oTd22);

  //añada la tabla al cuerpo del documento
  document.body.appendChild(oTable);:
```

Métodos de Tablas.

- Es un tanto compicado y difícil de seguir. Para facilitar la creación de tablas, el DOM HTML añade varias propiedades y métodos.
- El elemento <table /> añade lo siguiente:
  - tcaption: Puntero al elemnento <caption />
  - tBodies: Colección de elementos <tbody />
  - tFoot: Puntero al elemneto <thead />
  - tHead: Colección de todas las filas de la tabla.
  - createtHead: Crea un elemento <thead/> y lo añade a la tabla.
  - createTfoot(): Crea un elemento <tfoot /> y lo añade a la tabla.
  - createCaption(): Crea un elemento <caption /> y lo añade a la tabla después.
  - deleteTHead(): Borra el elemento <thead />.

Métodos de Tablas.

- Elemento <table> (continua):
  - deleteTFoot(): Borra el elemento <tfoot />.
  - deleteCaption(): Borra el elemento <caption />
  - **deleteRow**(posición): Elimina la fila en la posición indicada.
  - **insertRow** (posición): Añade una fila a la colección de filas en la posición indicada.
- El elemento <tbody /> añade lo siguiente:
  - **rows**: Colección de filas del elemento <tbody />.
  - **deleteRow**(posición): Elimina la fila en la posición indicada.
  - **insertRow**(posición): Añade una fila a la colección de filas en la posición indicada.

Métodos de Tablas.

- El elemento <tr /> añade lo siguiente:
  - **cells**: Colección de celdas del elemento <tr />.
  - **deleteCell**(posición): Elimina la celda en la posición indicada.
  - **insertCell**(posición): Añade una celda a la colección de celdas en la posición indicada.
- Con todas estas propiedades y métodos nos resultará mas sencilla la realización y manejo de tablas.

Métodos de Tablas.

- La primera parte donde creamos la tabla es exactamente igual.

```
//cree a tabla

  var oTable =document.createElement("table");
  oTable.setAttribute("border","1");
  oTable.setAttribute("width","100%");
  //cree el cuerpo
  var oTbody =document.createElement("tbody");
  oTable.appendChild(oTbody);
```

Métodos de Tablas.

Como se puede ver donde se emplean método y propiedades especificos del DOM HTML es a la hora de crear las filas:

```
//cree la primera fila
  oTbody.insertRow(0);
  oTbody.rows[0].insertCell(0);
  oTbody.rows[0].cells[0].appendChild(document.createTextNode("celda 1,1"));
  oTbody.rows[0].insertCell(1);
  oTbody.rows[0].cells[1].appendChild(document.createTextNode("celda 2,1"));
  //Cree la segunda fila
   oTbody.insertRow(1);
  oTbody.rows[1].insertCell(0);
  oTbody.rows[1].cells[0].appendChild(document.createTextNode("celda 1,2"));
  oTbody.rows[1].insertCell(1);
  oTbody.rows[1].cells[1].appendChild(document.createTextNode("celda 2,2"));
```

The End.