UD6: Modelo de Objetos del Documento

Antonio Sierra
Basado en el trabajo de Javier G. Pisano

Índice

Manipulación del DOM

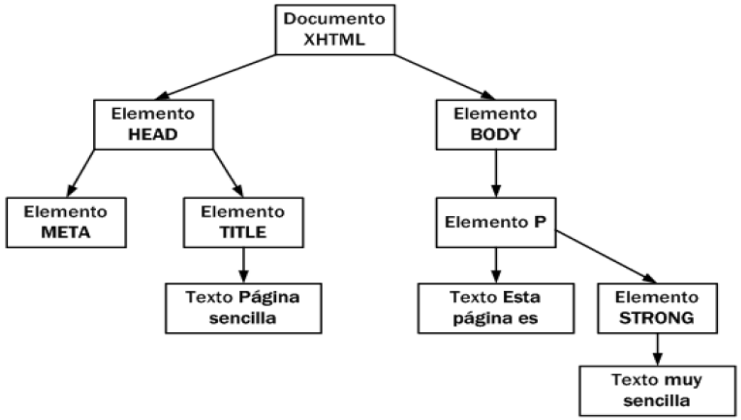
UD6: Modelo de Objetos del Documento

DOM (Document Object Model)

- Habitualmente utilizamos Javascript para acceder al contenido de la página Web.
- Desde Javascript puedo acceder a una estructura de datos que representa (en forma de árbol) todo el contenido de una página Web.
- Es posible:
  - Leer datos de la estructura
  - Modificar la estructura (añado contenido a la página Web).
    - Modificar elementos.
    - Añadir elementos.

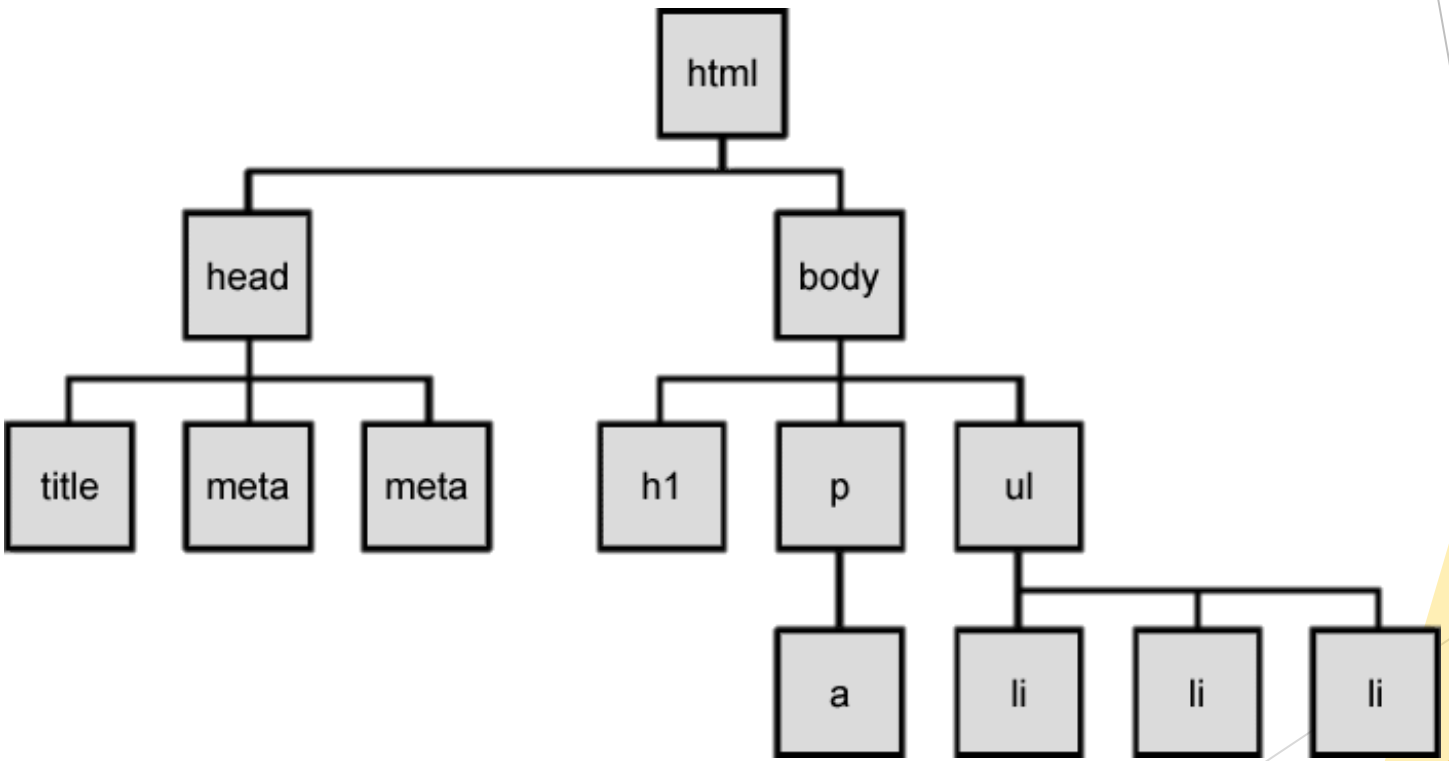
DOM: Ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Página sencilla</title>
</head>
<body>
    <p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```



DOM: Ejemplo

- ¿Qué elemento representa el siguiente árbol DOM?



Nodos del DOM (Node)

- Cada nodo del DOM es un objeto de un tipo derivado de la interfaz Node
- Existen 12 tipos de nodos, pero normalmente manejaremos los siguientes:

`Document`	Raíz del que derivan el resto. Además se instancia un objeto llamado **`document`** mediante el cual puedo acceder al contenido del documento.
`Element`	Etiqueta HTML. Puede contener atributos y otros nodos.
`Attr`	Atributos de las etiquetas
`Text`	Texto encerrado por una etiqueta HTML
`Comment`	Comentarios

Objeto document

- Representa el documento cargado en una ventana del navegador
- Permite el **acceso a todas las etiquetas HTML** dentro de una página
- Forma parte del objeto **window**, luego puede ser accedido mediante **window.document** o directamente **document**

Accediendo al DOM
Métodos de document

`getElementById(id)`	Devuelve el elemento con id **`id`**
`getElementsByClassName(class)`	Devuelve un array de elementos que tienen clase **`class`**
`getElementsByName(n)`	Devuelve un array de elementos que tienen el nombre (atributo **`name`**) **`n`**
`getElementsByTagName(tagname)`	Devuelve un array de elementos cuya etiqueta es **`tagname`**
`querySelector(query)`	Devuelve el primer elemento que concuerda con el grupo de **selectores CSS** especificados entre paréntesis
`querySelectorAll(query)`	Devuelve un array de elementos que concuerdan con el grupo de **selectores CSS** especificados entre paréntesis

Accediendo al DOM: Ejemplos

```
var elemento1 = document.getElementById("id-1");
var elementosClase1 =
        document.getElementsByClassName("clase-1");

/* Accedemos a cada elemento de manera secuencial */
for(var i=0;i<elementosClase1.length;i++){
    var elementoClase1=elementosClase1[i]; }

/* Equivale a la primera sentencia */
elemento1 = document.querySelector("#id-1");

/* Equivale a la segunda sentencia */
Elementos = document.querySelectorAll(".clase-1");

/* Obtendría elementos[0] */
 var elemento1 = document.querySelector(".clase-1");
```

Accediendo al DOM

- Una vez que hemos accedido a un elemento, también podemos acceder a los nodos relacionados con el mismo:
  - A los hijos del mismo con la propiedad **childNodes**.
  - A su padre con la propiedad **parentNode**
  - A sus hermanos con las propiedades **previousSibling** y **nextSibling**.

Accediendo al DOM: Ejemplos

```
var listado = document.getElementById("ul");
var hijosListado = listado.childNodes;
for(var i=0; i<hijosListado.length; i++)
{
    console.log(hijosListado[i].innerHTML);
}

/* Manera alternativa de recorrer los hijos */
var hijo = listado.firstChild /* listado.childNodes[0] */

while(hijo != null)
{
    console.log(hijo.innerHTML);
    hijo=hijo.nextSibling;
}
```

Interfaz Node
Propiedades

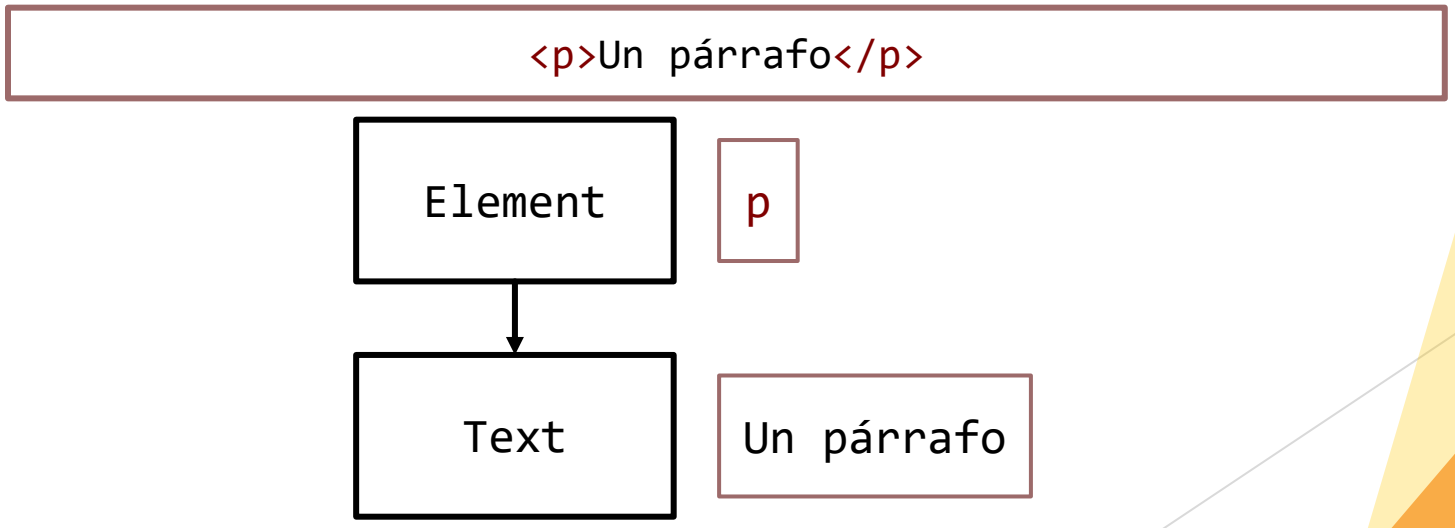
`childNodes`	Listado de nodos hijo del nodo actual
`firstChild`	Primer hijo del listado de nodos hijos.
`lastChild`	Último hijo del listado de nodos hijos.
`previousSibling`	Referencia al nodo hermano anterior o **`null`** si se trata del primer hijo
`nextSibling`	Referencia al nodo hermano siguiente o **`null`** si se trata del último hijo
`parentNode`	Referencia al padre del nodo actual
`attributes`	Atributos del nodo (solo para **`Element`**)

Interfaz Node
Métodos

`removeChild(nodo)`	Elimina el nodo cuya referencia le proporcionamos de la lista de nodos hijos.
`replaceChild(nuevo,reemplazado)`	Reemplaza el nodo viejo por el nuevo, siendo **`nuevo`** y **`reemplazado`** referencias a nodos.
`insertBefore(nuevo,anterior)`	Inserta el nodo **`nuevo`** antes de **`anterior`**, siendo ambos referencias a nodos.

Creación de elementos

- Un elemento HTML genera dos nodos:
  - Un nodo de tipo **Element** que representa la propia etiqueta
  - Un nodo de tipo **Text** que representa el texto de la etiqueta (lo que hay entre la apertura y el cierre de la misma.
    - No se genera si la etiqueta es de tipo sencillo (como **<img />**)



Creación de elementos

- La creación de un elemento consta de 4 pasos:
  - Creación del nodo de tipo **Element**
  - Creación del nodo de tipo **Text**
  - Añadir el nodo **Text** como nodo hijo del nodo **Element**
  - Añadir el nodo **Element** a la página, como hijo del nodo correspondiente al lugar donde quiero insertarlo.

Podíamos añadir el texto usando innerHTML, pero es más recomendable el método descrito
(por compatibilidad y rapidez)

Creación de elementos
Métodos de document

`createElement(tagname)`	Crea el elemento HTML cuya etiqueta indica **`tagname`**
`createTextNode(texto)`	Crea el nodo de texto con el contenido **`texto`**.
`createAttribute(name)`	Crea el atributo llamado **name** Posteriormente hay que agregarlo al elemento
`createComment(comentario)`	Crea el comentario con el contenido **`comentario`**.

Creación de elementos
Ejemplo

- Añadimos un párrafo al final de la página:

```
var parrafo = document.createElement("p");
var contenido =
        document.createTextNode("Contenido del párrafo");

parrafo.appendChild(contenido);
document.body.appendChild(parrafo);
```

Eliminación de nodos

- Para eliminar un nodo llamamos a la función **removeChild**() del nodo padre del que queremos borrar, pasando el propio nodo a borrar.

```
var padre=document.getElementById("listado")
var elementoBorrar=document.getElementById("borrame");

padre.removeChild(elementoBorrar);
```

Eliminación de nodos

- Lo más común es acceder al padre del elemento a borrar mediante la propiedad **parentNode** de éste.

```
var elementoBorrar=document.getElementById("borrame");

elementoBorrar.parentNode.removeChild(elementoBorrar);
```

Reemplazando nodos

- Podemos reemplazar un nodo llamando al método **replaceChild()**

```
/* Reemplazamos el primer elemento */
var primerHijo=listado.firstChild;

var nuevoHijo=document.createElement("li");
var texto=document.createTextNode("Nuevo texto");
nuevoHijo.appendChild(texto);

listado.replaceChild(nuevoHijo,primerHijo);
```

EJERCICIO PROPUESTO

- Crear html con un div que contenga un enlace y un párrafo con el texto: Me van a cambiar.
- Al cargar la página debe:
  - Mostrar mensaje que indique que iniciamos los cambios.
  - Añadir un párrafo al final del body con el texto: Contenido párrafo. Mostrar mensaje.
  - Mostrar mensaje indicando que se va a borrar el enlace, borrarlo y mostrar mensaje indicando que se borró.
  - Mostrar mensaje indicando que se va a cambiar el párrafo, cambiar el párrafo por otro cuyo contenido sea vuestro nombre y mostrar mensaje de que se cambió.
  - Añadir un párrafo con el mes actual antes del div.

Acceso a los atributos

- Podemos leer/escribir los atributos de los nodos de forma directa.
  - Los nodos tienen atributos que representan los atributos HTML
  - Accedo con el nombre del atributo en minúsculas.
  - El nombre coincide salvo en el caso del atributo XHTML **class**, al que accedo con **classNameName**.

```
<a id="enlace" href="http://www.agl.es">AGL</a>
```

```
var enlace=document.getElementById("enlace");
console.log(enlace.href); //muestra http://www.agl.es
```

Acceso a los atributos

- Otra alternativa para acceder a los atributos es usar los métodos:
  - elemento.getAttribute(nombre)
  - elemento.setAttribute(nombre,valor)
  - elemento.removeAttribute(nombre);

```
enlace.setAttribute("href","http://www.agl.es");
console.log(enlace.getAttribute("href"));
```

Acceso a los atributos

- También podemos acceder a todos los atributos de un elemento a través de su propiedad **attributes**

```
/* Acceso a todos los atributos de elemento */
var atributos=elemento.attributes;

for(var i=0;i<atributos.length;i++)
console.log(atributos[i].name+"-->"+atributos[i].value);
```

Nodos
Atributos

createAttribute	Crea un nodo tipo atributo nuevo con el nombre especificado en el parámetro.
getAttribute	Devuelve el valor del atributo con el nombre especificado para el elemento actual.
getAttributeNode	Devuelve el nodo tipo atributo con el nombre especificado del elemento actual.
getNamedItem	Devuelve el nodo tipo atributo con el nombre especificado de la colección actual de atributos.
hasAttribute	Devuelve si el elemento actual tiene un atrinuto con el nombre especificado o no.

Nodos
Atributos

`removeAttribute`	Elimina el atributo con el nombre especificado del elemento actual.
`removeAttributeNode`	Elimina el nodo tipo atributo con el nombre especificado del elemento actual.
`removeNamedItem`	Elimina el atributo con el nombre especificado de la lista de atributos actual y devuelve el nodo tipo atributo eliminado.
`setAttribute`	Añade un atributo a un elemento con el nombre y valor especificados.
`setAttributeNode`	Añade el nodo tipo atributo especificado al elemento actual.

Ejemplo crear atributos

```
<head>
    <script type="text/javascript">
        function CreateAttri(button) {
            var newAttr =
                    document.createAttribute("miAtributo");
            newAttr.value =
                    "Valor del atributo 'miAtributo'";
            button.setAttributeNode (newAttr);
            alert (button.getAttribute("miAtributo"));
        }
    </script>
</head>
<body>
    <button onclick="CreateAttri(this);">
                    Crea un atributo nuevo</button>
</body>
```

Ejemplo crear atributos

```
<head>
    <script type="text/javascript">
        function CreateAttri2(button) {
            button.setAttribute("miAtributo",
                        "Valor del atributo 'miAtributo'");
            alert(button.getAttribute("miAtributo"));
        }
        function ModificaEti(button) {
            button.setAttribute("value", "Etiqueta nueva");
        }
    </script>
</head>
<body>
    <button onclick="CreateAttri2(this);">
                        Crea un atributo nuevo</button>
    <input type="button" value="Modifica la etiqueta"
                        onclick="ModificaEti(this);" />
</body>
```

Nodos tipo comentario

```
function creaComentario() {
    var c = document.createComment("Mis comentarios");
    document.body.appendChild(c);
    var x = document.getElementById("parrafo");
    x.innerHTML = "Se anadio un comentario";
}
```

- Dentro del html en el body aparecerá:

```
<!--Mis comentarios-->
```

EJERCICIO PROPUESTO

- Crear html con div. Al cargar debe:
  - Crear mediante DOM 3 div, cada una de ellas con 3 párrafos. Añadir además a cada div
    un listado UL con 4 elementos li. Mediante bucles FOR.
  - Cada div debe tener como texto el número de división y como id div+nrodiv.
  - Cada párrafo debe tener como contenido Párrafo+nrodiv+.+nroparrafo
  - Cada elemento de la lista debe tener como contenido: Elemento de la lista + nrodiv+ . +nroelemlista

Acceso a la clase

- Propiedad elemento.className
  - Acceso a la clase del elemento
  - Si hay varias clases se separan con un espacio.
- Propiedad elemento.classList
  - Acceso a las clases del elemento como un listado.
  - Métodos para manejar el listado:
    - elemento.classList.add(clase [,clase]). Añade las clases indicadas, si ya existen se ignora.
    - elemento.classList.remove(clase). Elimina la clase indicada, si no existe se ignora.
    - elemento.classList.toggle(clase). Si la clase no existe se añade, y si existe se elimina.

Acceso al estilo

- También puedo acceder al estilo de los elementos a través de la propiedad **style**.
- La transformación de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio.
- Ejemplos:
  - font-weight se transforma en fontWeight
  - line-height se transforma en lineHeight.
  - border-top-style se transforma en borderTopStyle.
  - list-style-image se transforma en listStyleImage.

Acceso al estilo

- Mediante este procedimiento no se puede acceder a las propiedades de estilo definidas en el CSS.
- El procedimiento se usa para asignar y leer propiedades CSS in-line.

```
elemento.style.backgroundColor="red";
elemento.style.fontSize="1.5em";
elemento.style.backgroundImage=="url('fondo.png')";
```

Acceso al estilo

- En lugar de asignar todas las propiedades de CSS de manera individual, se recomienda usar la siguiente técnica:
  - Agrupar las propiedades CSS que queremos asignar en una clase definida en el archivo CSS
  - Asignar directamente la clase al elemento cuyo estilo queremos cambiar (o quitarla si queremos quitar el estilo).

```
elemento.className=“estilo-chulo";
```

```
estilo-chulo{
    background-color: "red";
    font-size: 1.5em;
    background-image: url("fondo.png");
}
```

Manejo de tablas desde el DOM

- DOM proporciona métodos específicos para trabajar con tablas, dado lo tedioso que resultaría trabajar con ellas de la manera vista hasta ahora

Sr. No	Roll No	Name	Team
1	1001	John	Red
2	1002	Peter	Blue
3	1003	Henry	Green

Elemento table
Propiedades y métodos

`rows`	Array con las filas de la tabla
`tBodies`	Array con todos los **`tbody`** de la tabla (Filas que no forman parte de la cabecera ni el pie de la tabla)
`insertRow(posicion)`	Inserta una nueva fila en la posición indicada
`deleteRow(posicion)`	Elimina una fila de la posición indicada

Elemento tr
Propiedades y métodos

`cells`	Devuelve un array con las columnas de la fila.
`insertCell(posicion)`	Inserta una nueva columna en la posición indicada
`deleteCell(posicion)`	Elimina la columna de la posición indicada

EJERCICIO PROPUESTO

- Crea 10 párrafos en la página desde un script, asignándoles un texto, un id diferenciado y una clase común para todos ellos.
- Crea un formulario que tenga:
  - Una caja de texto
  - Un botón Añadir. Al pulsar el mismo, se añadirá a la página HTML un elemento de lista cuyo contenido será el valor de la caja de texto. Si es la primera vez que se pulsa, deberá crearse el listado.
  - Un botón Quitar. Al pulsar el mismo, se eliminará del listado todos aquellos valores cuyo texto coincida con el valor introducido en la caja de texto.

EJERCICIO PROPUESTO

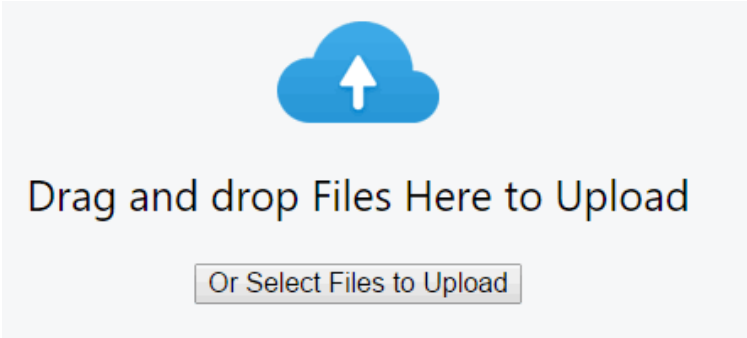
- Crea html con un div que contenga un párrafo con tu nombre. Después con javascript, añade una tabla de 5 filas con 3 columnas siendo el contenido de cada celda Celda + nrofila+ nrocol

Drag & Drop

UD6: Modelo de Objetos del Documento

Drag & Drop

- Mecanismo basado en eventos que nos permite crear contenido arrastrable
- API integrada en HTML5
  - Anteriormente había que utilizar librerías para lograr esta funcionalidad.



Drag & Drop Fundamentos

- El elemento que queremos arrastrar debe ser identificado como arrastrable (draggable).
  - Por defecto, todos los enlaces, imágenes y nodos de texto (incluidas las selecciones) son arrastrables.
- Desde JS escucharemos los eventos que se producen para producir la funcionalidad deseada.
  - Por defecto no hay acciones asociadas a los eventos de arrastrar sobre ningún elemento

Drag & Drop
Comprobando la funcionalidad

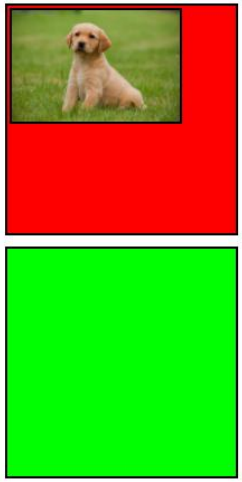
```
function compruebaDragAndDrop(){
    var div =document.createElement("div");
    return (('draggable' in div) ||
            ("ondragstart" in div && "ondrop" in div));
}
```

- También podemos usar la librería **Modernizr** que nos permite comprobar compatibilidad del navegador con muchas características

Drag & Drop
Creando contenido arrastrable

HTML

```
<div id="origen">
    <img id="perro" draggable="true"
         src="perro.jpg" /></div>
<div id="destino"></div>
```



CSS

```
[draggable] { user-select: none; }
#perro:hover {
  border: 2px dashed black;
  cursor: move;
}
```

Drag & Drop
Eventos de arrastre

dragstart	Comienza el arrastre. El target del evento hace referencia al elemento siendo arrastrado.
drag	El elemento se ha movido. El target hace referencia al elemento siendo movido. El evento se dispara tantas veces como se mueva el elemento.
dragenter	El elemento entra en un contenedor. El target hace referencia al contenedor
dragleave	El elemento sale de un contenedor. El target hace referencia al contenedor.

Drag & Drop
Eventos de arrastre

dragover	El elemento se mueve dentro del contenedor. El target hace referencia al contenedor. El comportamiento por defecto es denegar el drop, luego el evento debe devolver false o llamar a preventDefault del evento para indicar que se puede soltar.
drop	El elemento que se mueve ha sido exitosamente soltado en el contenedor. El target hace referencia al contenedor
dragend	Se ha dejado de arrastrar el elemento (con éxito o no). El target hace referencia al elemento arrastrado

Drag & Drop
Comienzo de la operación de arrastre

- Cambiamos la opacidad del elemento como indicativo de que comienza la operación
  - Tendremos que volver a fijar la transparencia de nuevo al 100% cuando finalice la operación.

CSS

```css
.transparente{
    opacity:0.4;}
```

JS

```js
var perro=document.getElementById("perro");
perro.addEventListener("dragstart",comienzaArrastre);

function comienzaArrastre(){
    this.classList.add("transparente");}
```

Drag & Drop
Transcurso de la operación de arrastre

- Es importante que en el manejador del evento dragover anulemos la acción por defecto, que es denegar el arrastre.
- Podemos utilizar el resto de eventos para ir dando pistas visuales de la operación de arrastre
  - Ejemplo: Cambiamos el borde del elemento de destino para indicar que podemos soltar el elemento

CSS

```
.sobre-contenedor{
    border:2px dashed black; }
```

Drag & Drop
Transcurso de la operación de arrastre

```
var destino=document.getElementById("destino");

destino.addEventListener("dragenter",entraContenedor);
destino.addEventListener("dragover",mueveEnContenedor);
destino.addEventListener("dragleave",fueraContenedor);

function entraContenedor(){
   this.classList.add("sobre-contenedor");}

function mueveEnContenedor(e){
  var evento=e || window.event;
  evento.preventDefault();
  /* Anulamos la acción por defecto, que es denegar el arra
stre */}

function fueraContenedor(){
  this.classList.remove("sobre-contenedor");}
```

Drag & Drop
Finalización de la operación de arrastre

- Para que se procese la operación de soltar, debemos añadir un manejador para el evento **dragend** en el objeto arrastrado.
  - En el manejador habrá que impedir el comportamiento predeterminado del navegador para este tipo de operaciones (ejemplo, mostrarlo si es una imagen).
  - Para ello, evitamos la propagación con **e.stopPropagation()**

Drag & Drop
Finalización de la operación de arrastre

```
destino.addEventListener("drop",suelta);
perro.addEventListener("dragend",finalizaArrastre);

function suelta(){
    this.classList.remove("sobre-contenedor");
}

function finalizaArrastre(e){
    var evento=e || window.event;
    evento.stopPropagation();
}
```

Drag & Drop
Objeto e.dataTransfer

- Es el centro de la actividad de Drag & Drop, pues es quien contiene los datos que se envían en la operación de arrastre.
  - Se establece en el manejador de evento dragstart
  - Se lee/procesa en el manejador de evento drop
- e.dataTransfer.setData(tipoMIME,dato)
  - Establezco el tipo de contenido del objeto y transmito la carga de datos.
  - Si no se establece explícitamente, contiene la etiqueta HTML sobre el cual se definió el dragstart
- e.dataTransfer.getData(tipoMIME,dato)
  - Obtiene los datos cargados

Drag & Drop
Objeto e.dataTransfer

```
function comienzaArrastre(e){
    var evento=e || window.event;
    this.classList.add("transparente");
    evento.dataTransfer.setData("text/html",
                                    evento.target.id);
}

 function suelta(e){
   var evento=e || window.event;
   this.classList.remove("sobre-contenedor");

   var id=evento.dataTransfer.getData("text/html");
   var elemento=document.getElementById(id);
   elemento.parentNode.removeChild(elemento);
   evento.target.appendChild(elemento);
}
```

Drag & Drop Resumiendo

- Elementos mínimos obligatorios

Elemento a arrastrar tiene la propiedad **draggable** con valor true

Evento **dragstart** sobre elemento a arrastrar definido

Establecemos el dato a enviar con **e.dataTransfer.setData()**

Evento **dragend** sobre elemento a arrastrar definido

Llamamos a **e.stopPropagation()**

Evento **drop** sobre elemento donde arrastramos definido

Accedemos al dato enviado con **e.dataTransfer.getData()**

Evento **dragover** sobre elemento donde arrastramos definido

Permitimos el arrastre con **e.preventDefault()**

File

UD6: Modelo de Objetos del Documento

File

- Es una API ofrecida por HTML5 para interactuar con archivos locales.
- Se puede utilizar para tratar archivos antes de enviarlos al servidor.
- Permite, entre otros:
  - Crear una vista previa en miniatura de las imágenes
  - Restringir el tamaño de un fichero antes de enviarlo al servidor
  - Verificar si el tipo MIME de un fichero coincide con los permitidos por el servidor

API File

- Se proporcionan las siguientes interfaces:

`File`	Representa un archivo local y proporciona información de lectura: Nombre, tamaño del archivo, tipo MIME y referencia al manejador de archivo
`FileList`	Representa un listado de objetos File. Pueden ser seleccionados a través de **<input type="file" multiple />** o a través de un conjunto de ficheros arrastrados desde el sistema (detectables mediante Drag and Drop usando el objeto **DataTransfer**).
`Blob`	Permite fragmentar un archivo en intervalos de *bytes*.
`FileReader`	Permite leer un archivo de forma asíncrona mediante el control de eventos de JS. Puede controlar el progreso de una lectura o detectar si se ha finalizado la carga de un fichero.

API File
Comprobando la funcionalidad

```
function compruebaFile(){
    if (window.File && window.FileReader && window.FileList

                                            && window.Blob)
        return true;
    else
        return false;
}
```

- También podemos usar la librería **Modernizr** que nos permite comprobar compatibilidad del navegador con muchas características

Blob

- Un objeto Blob representa un objeto tipo fichero de datos planos inmutables.
- Propiedades y métodos

`size`	Sólo lectura. Tamaño del archivo en bytes.
`type`	Sólo lectura. Tipo MIME del archivo.
`slice (inicio, fin, contentType);`	Devuelve un nuevo objeto Blob que contiene los datos en el rango de bytes especificado (todos los parámetros son opcionales).

File

- Propiedades
  - Al implementar **Blob** tiene las propiedades de éste.

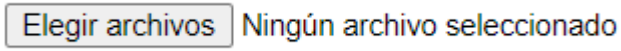
`lastModified`	**Sólo lectura**. Fecha/hora de la última modificación del archivo, expresada en formato UNIX (milisegundos desde 1 de enero de 1970).
`name`	**Sólo lectura**. Fecha/hora de la última modificación del archivo, expresada en formato UNIX (milisegundos desde 1 de enero de 1970).

Acceso a través de un formulario

- Agregamos un formulario donde usamos el atributo **multiple** para permitir la selección simultánea de archivos:

HTML

```
<input type="file" id="archivos" multiple />
<div id="lista-archivos"></div>
```



Acceso a través de un formulario

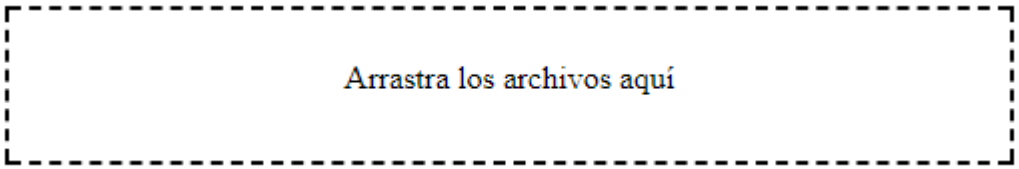
```
document.getElementById('archivos’).
                addEventListener('change', escogeArchivo);

function escogeArchivo(e){
    var archivos = e.target.files; // FileList

    var salida = "<ul>";
    for (var i = 0;i<archivos.length; i++) {
        salida+="<li>Nombre: "+archivos[i].name+" ";
        salida+="Tipo: "+archivos[i].type+" ";
        salida+="Tamaño: "+archivos[i].size+" bytes ";
        salida+="Última modificación: "+
                        new Date(archivos[i].lastModified).
                        toLocaleDateString()+" </li>";}
    salida+= "</ul>";

    document.getElementById("lista-archivos").
                                        innerHTML=salida;}
```

Acceso usando Drag & Drop

- Podemos modificar el ejemplo anterior para permitir arrastrar archivos desde el escritorio al navegador.
  - Algunos navegadores tratan los input de tipo **file** como destinos donde arrastrar archivos.

HTML

```
<div id="zona-arrastre">Arrastra los archivos aquí</div>
<div id="lista-archivos"></div>
```



Acceso usando Drag and Drop

- No es necesario definir los eventos sobre el elemento de origen
- **e.dataTransfer.files** contiene el listado de archivos arrastrados

```
var zonaArrastre = document.getElementById('zona-arrastre');
zonaArrastre.addEventListener('dragover', handleDragOver);
zonaArrastre.addEventListener('drop', handleDrop);

function handleDragOver(e){
    e.preventDefault();    e.stopPropagation();}

function handleDrop(e){
    e.preventDefault();    e.stopPropagation();
    var archivos = e.dataTransfer.files; // FileList

/* La composición de la salida es igual al ejemplo anterior*/
   document.getElementById("lista-archivos").innerHTML=salida;}
```

FileReader

- Una vez que hemos obtenido una referencia de File, podemos crear una instancia de FileReader para leer su contenido y almacenarlo en memoria.
  - Cuando finaliza la carga, se lanza el evento **onload** de **FileReader** y podemos acceder a la propiedad **result** para acceder a los datos del archivo.

FileReader

- FileReader ofrece 4 métodos para la lectura del archivo

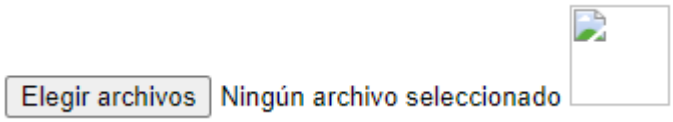
readAsBinaryString(Blob|File)	Obtiene los datos del archivo en forma de cadena binaria.
readAsText(Blob|File,encoding)	Obtiene los datos del archivo en forma de texto con la codificación indicada (por defecto UTF-8).
readAsDataURL(Blob|File)	Obtiene una URL que nos permite acceder a los datos del archivo.
readAsArrayBuffer(Blob|File)	Obtiene los datos del archivo como un objeto ArrayBuffer.

FileReader
Ejemplo

- Vamos a mostrar una vista previa del archivo, filtrando que se trate de un archivo de imagen.

HTML

```
<input type="file" id="archivos" multiple />
<img id="vista-previa" width="50" src="#"/>
```



FileReader
Ejemplo

```
document.getElementById('archivos’).
                addEventListener('change', escogeArchivo);

function escogeArchivo(e){
    var archivos = e.target.files; // FileList

    for (var i = 0;i<archivos.length; i++) {
         if(archivos[i].type.match('image.*')){
               var reader=new FileReader();
               reader.addEventListener("load",cargaImagen);
               reader.readAsDataURL(archivos[i]);}}}

function cargaImagen(e){
    document.getElementById("vista-previa").
                                        src=e.target.result;}
```

EJERCICIO PROPUESTO

- Modifica el método carga imagen para que cree la imagen y establezca los atributos.