Direcció Territorial d'Alacant
I.E.S. La Mar
Av. Augusta, 2. - 03730 Xàbia.
Tel.- 966428205 + Fax.- 966428206
Email: 03013339@edu.gva.es

Document Object Model (DOM)

- Document Object Model (DOM)
  - Introducción
  - Acceso a los nodos
  - Acceso a nodos a partir de otros
    - Propiedades de un nodo
  - Manipular el árbol DOM
    - Modificar el DOM con ChildNode
  - Atributos de los nodos
    - Estilos de los nodos
    - Atributos de clase

Introducción

La mayoría de las veces que programamos con Javascript es para que se ejecute en una página web mostrada por el navegador. En este contexto tenemos acceso a ciertos objetos que nos permiten interactuar con la página (DOM) y con el navegador (Browser Object Model, BOM).

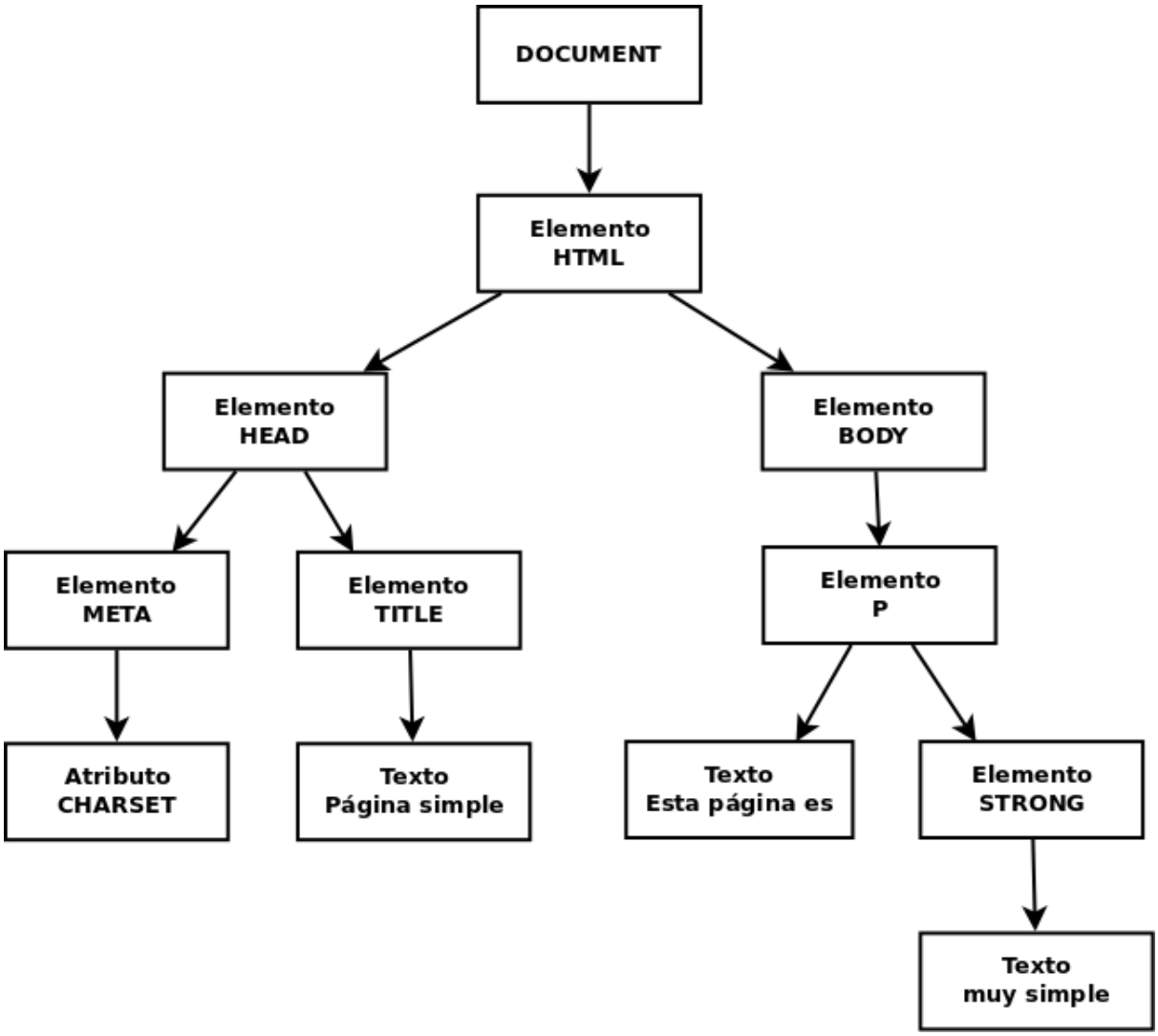
El **DOM** es una estructura en árbol que representa todos los elementos HTML de la página y sus atributos. Todo lo que contiene la página se representa como nodos del árbol y mediante el DOM podemos acceder a cada nodo, modificarlo, eliminarlo o añadir nuevos nodos de forma que cambiamos dinámicamente la página mostrada al usuario.

La raíz del árbol DOM es **document** y de este nodo cuelgan el resto de elementos HTML. Cada uno constituye su propio nodo y tiene subnodos con sus *atributos*, *estilos* y elementos HTML que contiene.

Por ejemplo, la página HTML:

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

se convierte en el siguiente árbol DOM:



Cada etiqueta HTML suele originar 2 nodos:

- **Element**: correspondiente a la etiqueta
- **Text**: correspondiente a su contenido (lo que hay entre la etiqueta y su par de cierre)

Cada nodo es un objeto con sus propiedades y métodos.

El ejemplo anterior está simplificado porque sólo aparecen los nodos de tipo ***elemento*** pero en realidad también generan nodos los saltos de línea, tabuladores, espacios, comentarios, etc.

Acceso a los nodos

Los principales métodos para acceder a los diferentes nodos son:

- **.getElementById(id)**: devuelve el nodo con la *id* pasada.
- **.getElementsByClassName(clase)**: devuelve una colección (similar a un array) con todos los nodos de la *clase* indicada. Ej.:

  NOTA: las colecciones son similares a arrays (se accede a sus elementos con *[indice]*) pero no se les pueden aplicar sus métodos *filter*, *map*, ... a menos que se conviertan a arrays con *Array.from()*

- **.getElementsByTagName(etiqueta)**: devuelve una colección con todos los nodos de la *etiqueta* HTML indicada.

- **getElementsByName(String name):** devuelve una lista de elementos con el nombre (atributo name), habitualmente se utilitza en formularios.
- **.querySelector(selector)**: devuelve el primer nodo seleccionado por el *selector* CSS indicado.
- **.querySelectorAll(selector)**: devuelve una colección con todos los nodos seleccionados por el *selector* CSS indicado.

Ejemplo1:

```
<h1>Primera capçalera</h1>
<ul>
  <li>Primer element de la primera llista</li>
  <li>Segon element de la primera llista</li>
  <li>Tercer element de la primera llista</li>
</ul>
<h1 class="secundari">Segona capçalera</h1>
<p id="contingut" class="secundari paragraf">Un paràgraf</p>
<ul>
  <li>Primer element de la segona llista</li>
  <li>Segon element de la segona llista</li>
  <li>Tercer element de la segona llista</li>
</ul>
```

```
let elementsSecundaris = document.getElementsByClassName('secundari');
console.log('Contingut dels elements amb classe="secundari":');
for (let i = 0; i < elementsSecundaris.length; i++) {
  console.log(elementsSecundaris[i]);
}
```

```
Contingut dels elements amb classe="secundari":
  <h1 class="secundari">Segona capçalera</h1>
  <p id="contingut" class="secundari paragraf">Un paràgraf</p>
> |
```

```
let elementsLlista = document.getElementsByTagName('li');
console.log('Contingut  dels elements de la llista (<li></li>):');
for (i = 0; i < elementsLlista.length; i++) {
  console.log(elementsLlista[i]);
}
```

```
Contingut  dels elements de la llista (<li></li>):
▼<li>
    ::marker
    "Primer element de la primera llista"
  </li>
▼<li>
    ::marker
    "Segon element de la primera llista"
  </li>
▼<li>
    ::marker
    "Tercer element de la primera llista"
  </li>
▼<li>
    ::marker
    "Primer element de la segona llista"
  </li>
▼<li>
    ::marker
    "Segon element de la segona llista"
  </li>
▼<li>
    ::marker
    "Tercer element de la segona llista"
  </li>
>
```

```
let elementContingut = document.getElementById('contingut');
console.log(elementContingut);
```

```
<p id="contingut" class="secundari paragraf">Un paràgraf</p>
> |
```

BUSCAR elementos con 'querySelector' 'querySelectorAll'.

Devuelve una colección con el primer o con todos los nodos seleccionados por el *selector* CSS indicado.

Entre los selectores que se pueden utilitzar estan:

- **#identificador**: que contenga id="identificador".
- **.nom_classe**: que contenga class="nom_classe".
- **element**: que sea un elemento con la etiqueta element.
- **[type="button"]**: elementos que tengan el atributo type y su valor sea button.
- **:first-child** :elementos que son el primer hijo de cualquier otro elemento.
- **element:first-child** : elementos que son el primer hijo del elemento padre.

EJEMPLO SELECTOR

```
h1>Primera capçalera</h1>
<ul>
  <li>Primer element de la primera llista</li>
  <li>Segon element de la primera llista</li>
  <li>Tercer element de la primera llista</li>
</ul>
<h1 class="secundari">Segona capçalera</h1>
<p id="contingut" class="secundari paragraf">Un paràgraf</p>
<ul>
  <li>Primer element de la segona llista</li>
  <li>Segon element de la segona llista</li>
  <li>Tercer element de la segona llista</li>
</ul>
```

```
console.log('-- Capçaleras h1 amb classe secundari --');
let elementsTitols = document.querySelectorAll('h1.secundari');
for (let i=0; i<elementsTitols.length; i++) {
  console.log(elementsTitols[i].textContent);
}
```

```
-- Capçaleras h1 amb classe secundari --
Segona capçalera
```

```
console.log('-- Tots els elements de les llistes --');
let elementsLlista = document.querySelectorAll('li');

for (let i=0; i<elementsLlista.length; i++) {
  console.log(elementsLlista[i].textContent);
}
```

```
-- Tots els elements de les llistes --
Primer element de la primera llista
Segon element de la primera llista
Tercer element de la primera llista
Primer element de la segona llista
Segon element de la segona llista
Tercer element de la segona llista
```

```
console.log('-- Primers elements de les llistes --');
let elementsPrimers = document.querySelectorAll('li:first-child');
for (let i=0; i<elementsPrimers.length; i++) {
  console.log(elementsPrimers[i].textContent);
```

```
-- Primers elements de les llistes --
Primer element de la primera llista
Primer element de la segona llista
> |
```

También tenemos 'atajos' para obtener algunos elementos comunes:

Ejercicio: Probad los métodos siguientes con el fichero del ejemplo1.

- **`document.documentElement`**: devuelve el nodo del elemento <*html*>
- **`document.head`:** devuelve el nodo del elemento <*head*>
- **`document.body`:** devuelve el nodo del elemento <*body*>
- **`document.title`:** devuelve el nodo del elemento <*title*>
- **`document.link`**: devuelve una colección con todos los hiperenlaces del documento
- **`document.form`**: devuelve una colección con todos los formularios del documento
- **`document.images`:** devuelve una colección con todas las imágenes del documento
- **`document.scripts`:** devuelve una colección con todos los scripts del documento

Acceso a nodos a partir de otros

En muchas ocasiones queremos acceder a cierto nodo a partir de uno dado. Para ello tenemos los siguientes métodos que se aplican sobre un elemento del árbol DOM:

EJEMPLO 2:

** Primer paràgraf Segon paràgraf Tercer paràgraf **

- **`elemento.parentNode`**: devuelve el Nodo padre de *elemento*

  ```
  let padre = document.getElementById("primer");
  console.log(padre.parentNode);
  ```

  ```
  ▶ <div id="contenidor">…</div>
  > |
  ```

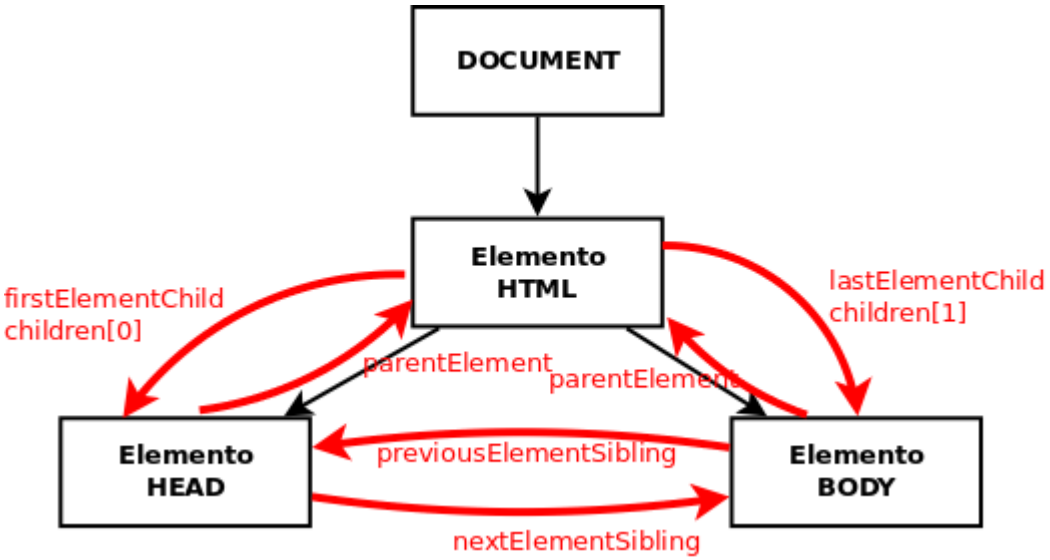
- **`elemento.firstChild`**: devuelve el nodo que es el primer hijo de *elemento* (incluyendo nodos de tipo texto o comentarios)
- **`elemento.lastChild`**: igual pero con el último hijo.

  ```
  let contenidor = document.getElementById("contenidor");
  console.log(contenidor.firstChild.textContent);
  console.log(contenidor.lastChild.textContent);
  ```

  ```
  Primer paràgraf
  Tercer paràgraf
  >
  ```

Ejercicio: Probar los siguiente métodos con el fichero del Ejemplo 2.

- **`elemento.nextSibling`**: devuelve el nodo que es el siguiente hermano de *elemento* (incluyendo nodos de tipo texto o comentarios)
- **`elemento.previousSibling`:** igual pero con el hermano anterior



- **`elemento.ChildNodes:`** Accedemos a la lista de Nodos que contiene.Podemos saber la cantidad de nodos que contiene con la propiedat length.
- **`elemento.hasChildNodes`**: indica si *elemento* tiene o no nodos hijos

Como la lista de nodos puede ser un *array*, se puede recorrer como tal.

EJEMPLO 3:

```
<div id="contenidor">
  <p id="primer">Primer paràgraf</p>
  <p id="segon">Segon paràgraf</p>
  <p id="tercer">Tercer paràgraf</p>
</div>
```

```
let contenidor = document.getElementById("contenidor");
console.log("El contenedor tiene otros nodos: ?" + contenidor.hasChildNodes());

console.log("Tiene: "  + contenidor.childNodes.length + "nodos.");

for (let i = 0; i < contenidor.childNodes.length; i++) {
 console.log("Posicion: " + i + " - Nodo: " + contenidor.childNodes[i].textContent);
}
```

```
El contenedor tiene otros nodos: ?true
Tiene: 7nodos.
Posicion: 0 - Nodo:

Posicion: 1 - Nodo: Primer paràgraf
Posicion: 2 - Nodo:

Posicion: 3 - Nodo: Segon paràgraf
Posicion: 4 - Nodo:

Posicion: 5 - Nodo: Tercer paràgraf
Posicion: 6 - Nodo:
>
```

Propiedades de un nodo

Las principales propiedades de un nodo son:

- **`elemento.innerHTML`**: todo lo que hay entre la etiqueta que abre *elemento* y la que lo cierra, incluyendo otras etiquetas HTML.

  Por ejemplo si *elemento* es el nodo `<p>Esta página es <strong>muy simple</strong></p>`

```
let contenido = elemento.innerHTML;   // contenido='Esta página es <strong>muy simple</strong>'
```

NOTA: Con esta propiedad podemos introducir texto en la pagina web.
Ejemplo: Ver_en_pantalla.

<div id="informacion"

<script>
let info = document.getElementById("informacion"); //Obtengo el elemento "informacion".
info.innerHTML = "Hola Mundo" //Le añado los datos que quiero se salgan por el Interface
</script>

- **`elemento.textContent`:** todo lo que hay entre la etiqueta que abre *elemento* y la que lo cierra, pero ignorando otras etiquetas HTML. Siguiendo con el ejemplo anterior:

```
let contenido = elemento.textContent;   // contenido='Esta página es muy simple'
```

- **`elemento.value`:** devuelve la propiedad 'value' de un <input> (en el caso de un <input> de tipo text devuelve lo que hay escrito en él). Como los <inputs> no tienen etiqueta de cierre (</input>) no podemos usar *.innerHTML* ni *.textContent*.

EJEMPLO 4:

```
<input name="nombre">Nombre<br>
<input type="radio" name="sexo" value="H">Hombre<br>
<input type="radio" name="sexo" value="M">Mujer
```

```
let elem1=document.getElementsByName('nombre');
console.log(elem1.value);

let elem2=document.getElementsByName('sexo');
let tam=elem2.length;
for (var i=0;i<tam;i++){
  console.log(elem2[i].value);
}
```

```
undefined
H
M
>
```

Ejercicio 5: Realizar el anterior ejercicio utilizando **querySelector** y **querySelectorAll**

```
let yo=document.querySelectorAll('input');
for(let i=1;i<yo.length;i++){
  console.log(yo[i].value);
}
```

Otras propiedades:

- **`elemento.innerText`:** igual que *textContent*
- **`elemento.focus`:** da el foco a *elemento* (para inputs, etc). Para quitarle el foco `elemento.blur`
- **`elemento.clientHeight` / `elemento.clientWidth`**: devuelve el alto / ancho visible del *elemento*

- **`elemento.offsetHeight / elemento.offsetWidth`:** devuelve el alto / ancho total del *elemento*
- **`elemento.clientLeft / elemento.clientTop`:** devuelve la distancia de *elemento* al borde izquierdo / superior
- **`elemento.offsetLeft / elemento.offsetTop`:** devuelve los píxels que hemos desplazado *elemento* a la izquierda / abajo

Manipular el árbol DOM

Vamos a ver qué métodos nos permiten cambiar el árbol DOM, y por tanto modificar la página:

- **`document.createElement('etiqueta')`:** crea un nuevo elemento HTML con la etiqueta indicada, pero aún no se añade a la página. Ej.:

```
let nuevoLi = document.createElement('li');
```

- **`document.createTextNode('texto')`:** crea un nuevo nodo de texto con el texto indicado, que luego tendremos que añadir a un nodo HTML. Ej.:

```
let textoLi = document.createTextNode('Nuevo elemento de
lista');
```

- **`elemento.appendChild(nuevoNodo)`:** añade *nuevoNodo* como último hijo de *elemento*. Ahora ya se ha añadido a la página. Ej.:

```
nuevoLi.appendChild(textoLi);     // añade el texto creado al
elemento LI creado
let miPrimeraLista = document.getElementsByTagName('ul')[0];
// selecciona el 1º UL de la página
miPrimeraLista.appendChild(nuevoLi);    // añade LI como último
hijo de UL, es decir al final de la lista
```

- **`elemento.insertBefore(nuevoNodo, nodo)`:** añade *nuevoNodo* como hijo de *elemento* antes del hijo *nodo*. Ej.:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];
// selecciona el 1º UL de la página
let primerElementoDeLista =
miPrimeraLista.getElementsByTagName('li')[0];   // selecciona
el 1º LI de miPrimeraLista
miPrimeraLista.insertBefore(nuevoLi, primerElementoDeLista);
// añade LI al principio de la lista
```

- **`elemento.removeChild(nodo)`:** borra *nodo* de *elemento* y por tanto se elimina de la página. Ej.:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];
// selecciona el 1º UL de la página
let primerElementoDeLista =
miPrimeraLista.getElementsByTagName('li')[0];  // selecciona el
1º LI de miPrimeraLista
miPrimeraLista.removeChild(primerElementoDeLista);    // borra
el primer elemento de la lista
```

- **`elemento.replaceChild(nuevoNodo, viejoNodo)`:** reemplaza *viejoNodo* con *nuevoNodo* como hijo de *elemento*. Ej.:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];
// selecciona el 1º UL de la página
let primerElementoDeLista =
miPrimeraLista.getElementsByTagName('li')[0];  // selecciona el
1º LI de miPrimeraLista
miPrimeraLista.replaceChild(nuevoLi, primerElementoDeLista);
// reemplaza el 1º elemento de la lista con nuevoLi
```

- **`elementoAClonar.cloneNode(boolean)`:** devuelve un clon de *elementoAClonar* o de *elementoAClonar* con todos sus descendientes según le pasemos como parámetro *false* o *true*. Luego podremos insertarlo donde queramos.

OJO: Si añado con el método **`appendChild`** un nodo que estaba en otro sitio **se elimina de donde estaba** para añadirse a su nueva posición. Si quiero que esté en los 2 sitios deberé clonar el nodo y luego añadir el clon y no el nodo original.

Ejercicio 6 : Utilizar el fichero del **Ejercicio1** para probar los ejemplos anteriores.

PRACTICA 5.

Modificar el DOM con ChildNode

Childnode es una interfaz que permite manipular del DOM de forma más sencilla.

Con Versiones antiguas de Safari no funciona.

- **`elemento.before(nuevoNodo):`** añade el *nuevoNodo* pasado antes del nodo *elemento*
- **`elemento.after(nuevoNodo)`**: añade el *nuevoNodo* pasado después del nodo *elemento*
- **`elemento.replaceWith(nuevoNodo):`** reemplaza el nodo *elemento* con el *nuevoNodo* pasado
- **`elemento.remove():`** elimina el nodo *elemento*

Atributos de los nodos

Podemos ver y modificar los valores de los atributos de cada elemento HTML y también añadir o eliminar atributos:

- **`elemento.attributes:`** devuelve un array con todos los atributos de *elemento*
- **`elemento.hasAttribute('nombreAtributo')`**: indica si *elemento* tiene o no definido el atributo *nombreAtributo*
- **`elemento.getAttribute('nombreAtributo'):`** devuelve el valor del atributo *nombreAtributo* de *elemento*. Para muchos elementos este valor puede directamente con **`elemento.atributo.`**
- **`elemento.setAttribute('nombreAtributo', 'valor'):`** establece *valor* como nuevo valor del atributo *nombreAtributo* de *elemento*. También puede cambiarse el valor directamente con **`elemento.atributo=valor.`**
- **`elemento.removeAttribute('nombreAtributo'):`** elimina el atributo *nombreAtributo* de *elemento*

A algunos atributos comunes como **`id`**, **`title`** o **`className`** (para el atributo **`class`**) se puede acceder y cambiar como si fueran una propiedad del elemento (`elemento.atributo`). Ejemplos:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];  // selecciona el 1º UL de la página
miPrimeraLista.id = 'primera-lista';
// es equivalente ha hacer:
miPrimeraLista.setAttribute('id', 'primera-lista');
```

Estilos de los nodos

Los estilos están accesibles como el atributo **style**.

Por ejemplo para cambiar el color de fondo (propiedad backgroundColor) y ponerle el color *rojo* al elemento *miPrimeraLista* haremos:

```
miPrimeraLista.style.backgroundColor = 'red';
```

De todas formas normalmente **NO CAMBIAREMOS ESTILOS** a los elementos sino que les pondremos o quitaremos clases que harán que se le apliquen o no los estilos definidos para ellas en el CSS.

Atributos de clase

Ya sabemos que el aspecto de la página debe configurarse en el CSS por lo que no debemos aplicar atributos *style* al HTML. En lugar de ello les ponemos clases a los elementos que harán que se les aplique el estilo definido para dicha clase.

Como es algo muy común en lugar de utilizar las instrucciones de **`elemento.setAttribute('className', 'destacado')`** o directamente **`elemento.className='destacado'`** podemos usar la propiedad ***classList*** que devuelve la colección de todas las clases que tiene el elemento. Por ejemplo si *elemento* es **`<p class="destacado direccion">...:`**

```
let clases=elemento.classList;   // clases=['destacado',
'direccion'], OJO es una colección, no un Array
```

Además dispone de los métodos:

- **.add(clase)**: añade al elemento la clase pasada (si ya la tiene no hace nada). Ej.:

  ```
  elemento.classList.add('primero');   // ahora elemento será
  <p class="destacado direccion primero">...
  ```

- **.remove(clase)**: elimina del elemento la clase pasada (si no la tiene no hace nada). Ej.:

  ```
  elemento.classList.remove('direccion');   // ahora elemento
  será <p class="destacado primero">...
  ```

- **.toogle(clase)**: añade la clase pasada si no la tiene o la elimina si la tiene ya. Ej.:

```
elemento.classList.toogle('destacado');   // ahora elemento será <p class="primero">...
elemento.classList.toogle('direccion');   // ahora elemento será <p class="primero direccion">...
```

- **.contains(clase)**: dice si el elemento tiene o no la clase pasada. Ej.:

```
elemento.classList.contains('direccion');   // devuelve true
```

- **.replace(oldClase, newClase)**: reemplaza del elemento una clase existente por una nueva. Ej.:

```
elemento.classList.replace('primero', 'ultimo');   // ahora elemento será <p class="ultimo direccion">...
```