# UD6. Utilización del Modelo de Objetos del Documento (DOM)

DWEC
Mª Luz Sánchez Rubio

# OBJETIVOS

- Reconocer el modelo de objetos del documento de una página web, identificando sus objetos, propiedades y métodos.
- Comprender y utilizar selectores avanzados de CSS.
- Manipular las clases para implementar efectos visuales y de interacción.
- Modificar las estructura del DOM dinámicamente mediante la creación, inserción y eliminación de elementos.
- Navegar la estructura del DOM mediante Traversing.
- Utilizar jQuery para manejar el DOM y el manejo de eventos.
- Usar JSDoc para documentar el código y  mejorar su claridad y mantenimiento.

# 1.- Introducción al DOM

- **DOM** es el acrónimo de **Document Object Model** (Modelo de objeto del Documento).
- Es una interfaz de programación que los navegadores generan a partir de un documento HTML o XML.
- Su función principal es representar la estructura del documento como un **árbol jerárquico** de nodos.
- Permite que los lenguajes de programación –JavaScript- puedan leerlo y modificarlo dinámicamente.
- El DOM se considera un **estándar del W3C**, lo que significa que su funcionamiento está definido de forma oficial y los navegadores lo implementan siguiendo estas directrices.

# 1.- Introducción al DOM

## 1.1.- DOM estático vs DOM vivo

- El **DOM estático** es la estructura inicial generada al cargar la página.

- El **DOM vivo** es la versión actualizada que cambia constantemente a media que JavaScript modifica elementos, añade o elimina nodos, o se re-renderiza parte de la interfaz.

# 1.- Introducción al DOM

## 1.2.- Tipos de nodos principales

- El **DOM** está compuesto por diferentes tipos de nodos, entre los que destacan.
  - **Nodo del Documento:** Representa el documento completo.
  - **Nodos de elemento**: Cada etiqueta HTML (ejemplo: , <p>).
  - **Nodos de texto**: Contienen solo el texto dentro de las etiquetas.
  - **Nodos de atributo**: Describen propiedades de los elementos.
  - **Nodos del comentario**: Representan comentarios <!-- -->.
- Cada uno cumple una función específica en la representación estructural del documento.

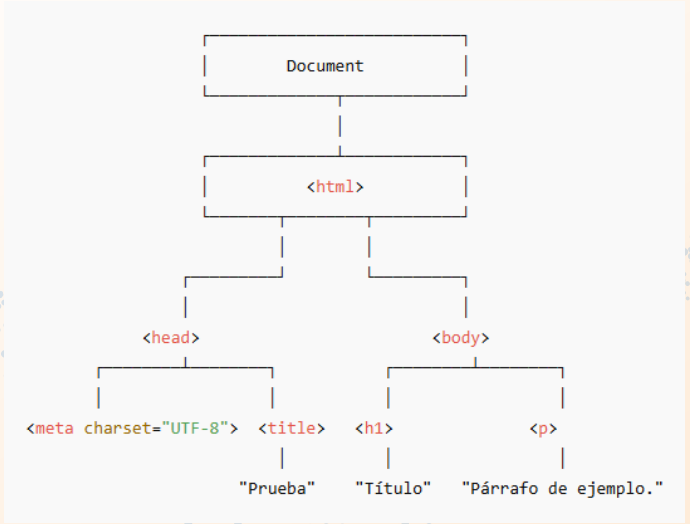
# 2.- Estructura del DOM como árbol

- El documento HTML se organiza en un **árbol de nodos** donde cada nodo representa una parte del documento (etiquetas, texto, atributos).
- Este árbol tiene un nodo raíz <html>, del cual descienden múltiples niveles de jerarquía.
- Relaciones jerárquicas:
  - **Padre**: Un nodo superior que contiene a otros nodos.
  - **Hijo**: Un nodo contenido por otro.
  - **Hermanos**: Nodos que comparten el mismo padre.

# 2.- Estructura del DOM como árbol

- Ejemplo:

```
<html>
 <head>
     <meta charset="UTF-8">
     <title>Prueba</title>
 </head>
 <body>
  <h1>Título</h1>
  <p>Párrafo de ejemplo.</p>
 </body>
</html>
```



# 2.- Estructura del DOM como árbol

- Relación de parentesto (padre, hijo, hermanos):
  - **Padre-hijo**
    - Document -> HTML
    - HTML -> Head y Body
    - Head -> Meta y Title
    - Body -> h1 y p
  - **Hermanos**
    - Head -> Body
    - Meta -> Title
    - h1 -> p

# 3.- ¿Cómo funciona el DOM?

- **Proceso de conversión**

1. El navegador descarga el documento HTML.
2. Se analiza línea por línea.
3. Se construye el árbol DOM.
4. Se habilita el acceso al DOM desde JavaScript.

# 4.- Propósito del DOM

- **Interacción dinámica**
  - ✓ Permite a JavaScript modificar la página sin necesidad de recargarla, esencial para aplicaciones modernas.

- **Manipulación del contenido**
  - ✓ Modificar el texto, añadir y eliminar elementos y modificar atributos de los elementos.

- **Control de estilos**
  - ✓ Modificar clases, estilos en línea, y aplicar cambios de diseño sobre la marcha.

# 5.- Selección avanzada con CSS en el DOM

- **querySelector(selector)**
  - ✓ Selecciona **el primer elemento** que coincide con un selector CSS (clase, id, nombre de etiqueta, etc.)
  - ✓ Ejemplo:

```
document.querySelector('.clase');
```

- **querySelectorAll(selector)**
  - ✓ Selecciona **todos los elementos** que coinciden con un selector CSS, devolviendo una **NodeList** (similar a un array, aunque no tiene todos los métodos de un array)
  - ✓ Ejemplo:

```
document.querySelectorAll('div');
```

# 6.- Selectores CSS comunes

- **Tipos de selectores en *querySelector* y *querySelectorAll***

  - ✓ Selector por **etiqueta**: ‘p’, ‘h1’, ‘div’
  - ✓ Selector por **clase**: ‘.miClase’
  - ✓ Selector por **ID**: ‘#miId’
  - ✓ Selectores **combinados**: ‘.miClase h1’, ‘#menu > li’
  - ✓ Selectores **atributos**: ‘input[type=“text”]’, ‘:first-child’, ‘:not(.activo)’

# 7.- Comparación entre selectores

| Método | Selección Única o Múltiple | Tipo de Retorno | Selector Admitido | Ejemplo Uso |
|---|---|---|---|---|
| `querySelector` | Única | Nodo | CSS Selector (complejo) | `.clase`, `#id`, `tag` |
| `querySelectorAll` | Múltiple | NodeList | CSS Selector (complejo) | `.clase`, `#id` `.subclase` |
| `getElementById` | Única | Nodo | Solo `id` | `#id` |
| `getElementsByTagName` | Múltiple | HTMLCollection | Solo etiquetas | `div`, `p` |
| `getElementsByName` | Múltiple | NodeList | Solo atributo `name` | `name="miInput"` |
| `getElementsByClassName` | Múltiple | HTMLCollection | Solo clase | `.clase` |

# 8.- Introducción a la manipulación de clases

- **¿Por qué manipular clases en el DOM?**
  - ✓ Permite cambiar el estilo de los elementos dinámicamente.
  - ✓ Ayuda a crear interacciones como mostrar/ocultar elementos, cambiar colores, aplicar animaciones, etc.
- **Herramientas clave:**
  - ✓ className
  - ✓ classList

# 8.- Introducción a la manipulación de clases

- **¿Qué es className?**
  - ✓ **className** es una propiedad que contiene todas las clases de un elemento como una cadena de text.
  - ✓ Modificar **className** reemplaza todas las clases actuales del elemento.

# 8.- Introducción a la manipulación de clases

- **Sintaxis**

  *elemento.className;*

- **Ejemplo:**

```
const miDiv = document.querySelector('#miDiv');
miDiv.className = 'nueva-clase';
//asignación múltiple de clases
miDiv.className = 'clase1 clase2 clase3';
//limpiar todas las clases
 miDiv.className = '';
```

# 8.- Introducción a la manipulación de clases

- **¿Qué es classList?**
  - ✓ **classList** es una propiedad que devuelve una lista activa de las clases que tiene un elemento
  - ✓ Es muy útil porque proporciona métodos específicos para manipular clases.
- **Sintaxis:**

  elemento.classList;

# 8.- Introducción a la manipulación de clases

- **Ejemplo:**

```
const miElemento = document.querySelector('.mi-elemento');
console.log(miElemento.classList); // Muestra las clases actuales del elemento
```

# 8.- Introducción a la manipulación de clases

- **Métodos de classList**
  - ✓ **add():** Añade una o más clases al elemento.

    *miElemento.classList.add('nueva-clase');*
  - ✓ **remove()**: Elimina una o más clases del elemento.

    *miElemento.classList.remove('nueva-clase');*
  - ✓ **toggle()**: Añade la clase si no existe y la elimina si está presente.

    *miElemento.classList.toggle('activo');*
  - ✓ **contains():** Verifica si el elemento contiene uma class específica

    *miElemento.classList.contains('activo'); // true o false*

# 8.- Introducción a la manipulación de clases

- **className vs classList**
  - ✓ **classList:**
    - Proporciona métodos específicos para agregar, eliminar y verificar clases individuales sin afectar otras clases.
    - Es ideal para manipular clases individuales sin afectar otras.
  - ✓ **className**:
    - Útil para establecer todas las clases de un elemento de una solo vez.
    - Modificar **className** reemplaza cualquier clase del elemento

# 9.- Manipulación de nodos en el DOM

- Modificar la estructura del DOM implica crear, insertar, eliminar o mover nodos.
- Crear Nodos en el DOM
  - ✓ Crea nodos de tipo elemento. **document.createElement()**

    **Ejemplo:** *const nuevoElemento=document.createElement("div")*

  - ✓ Crea nodos de texto. **document.createTextNode()**

    **Ejemplo:** *const texto=document.createTextNode("Hola mundo")*

- Los elementos creados deben agregarse a un elemento padre existente.

# 9.- Manipulación de nodos en el DOM

- **Añadir nodos al DOM**
  - ✓ **appendChild():** este método se usa para añadir un nodo al final de un nodo padre. Solo permite añadir nodos.

**Ejemplo:** *elementoPadre.**appendChild**(nuevoElemento).*

# 9.- Manipulación de nodos en el DOM

- **Añadir nodos al DOM**

**Código de ejemplo:**

```
const nuevoElemento = document.createElement("p");
const texto = document.createTextNode("Texto de ejemplo");
nuevoElemento.appendChild(texto);
document.body.appendChild(nuevoElemento);
```

- Crea un elemento de tipo <p>.
- Crea un nodo de texto puro que contiene la cadena “Texto de ejemplo”.
- Añade el texto al nodo **nuevoElemento,** pero aún no se ve en la página.
- Añade el **nuevoElemento** al **final del body.** Ahora ya se muestra en la página.

# 9.- Manipulación de nodos en el DOM

- **Añadir nodos al DOM**
  - ✓ Uso de **append().** Este método es más **moderno**, **flexible** y **más cómoda** de insertar contenidos dentro de DOM. Inserta uno o varios nodos o textos al final del elemento.

**Ejemplo:** *elementoPadre.**append**(nuevoEle);*
*elementoPadre.**append**(nuevoEle1, nuevoEle2, "nodo");*

  - ✓ **Código de ejemplo:**

```
const nuevoElemento = document.createElement("p");
nuevoElemento.textContent = "Este es un párrafo";
div.append("Texto adicional", nuevoElemento);
```

# 9.- Manipulación de nodos en DOM

- **append() vs appendChild()**

  - ✓ **append()** es más flexible, ya que puede insertar tanto nodos, como texto y permite múltiples argumentos en una sola llamada.
  - ✓ **appendChild()** es más antiguo y universalmente compatible, mientras que **append()** es más moderno y puede no estar soportado en navegadores más antiguos.
  - ✓ **appendChild**() devuelve el nodo insertado, mientras que **append()** no devulve nada
  - ✓ Ambos métodos son útiles, pero **append()** es preferido cuando se necesita mayor flexibilidad en la inserción de contenido.

# 9.- Manipulación de nodos en DOM

- **Otros métodos modernos para añadir elementos:**

  - ✓ **prepend():** inserta al inicio del elemento.

    lista.**prepend**(nodo)
  - ✓ **before() / after():** inserta nodos sin acceder al elemento padre

    lista.**before**(nodoNuevo)
  - ✓ **insertBefore()**: inserta un nodo antes de otro existente dentro del mismo padre.

    lista.**insertBefore**(nodoNuevo, nodoRef);

# 9.- Manipulación de nodos en DOM

- **Otros métodos modernos para añadir elementos:**
  - ✓ **insertAdjacentElement():** inserta un elemento nodo directamente en una posición concreta alrededor de un elemento sin borrar ni reconstruir el contenido existente

    lista.**insertAdjacentElement**(**posicion**, elemento)
  - ✓ **insertAdjacentHTML():** inserta HTML

    lista.**insertAdjacentHTML**(**posicion**, html)

posición:

- "beforebegin"-> Justo antes del elemento.
- "afterbegin" -> Dentro del elemento, al inicio.
- "beforeend" -> Dentro del elemento, al final.
- "afterbegin" -> Justo después del elemento

lista.**insertAdjacentHTML**("beforeend", " Hola ");

# 9.- Manipulación de nodos en DOM

- **Eliminar nodos**

  ✓ Uso de **removeChild().** Este método elimina un nodo hijo desde un nodo padre.

  **Ejemplo:** *elementoPadre.removeChild(elementoEliminar)*

  ✓ **Código ejemplo**

```
const elementoPadre = document.querySelector(“#contenedor");
const nodoAEliminar = document.querySelector(“#elementoHijo");
elementoPadre.removeChild(nodoAEliminar);
```

# 9.- Manipulación de nodos en DOM

- **Eliminar nodos**

  ✓ Uso de **remove**() este método es más moderno y permite eliminar un nodo directamente sin necesidad de hacer referencia a su elemento padre:

  **Ejemplo:** *elementoEliminar.remove();*

  ✓ **Código ejemplo:**

```
const nodoAEliminar = document.getElementById("elementoHijo");
nodoAEliminar.remove();
```

# 9.- Manipulación de nodos en DOM

- **Reemplazar nodos**

✓ Uso de **replaceChild**() este método es utilizado para reemplazar un nodo hijo existente con un nuevo nodo dentro de un nodo padre.

**Ejemplo:** *elementoPadre.**replaceChild**(nuevoNodo, nodoExistente);*

✓ **Código ejemplo:**

```
let nuevoElemento = document.createElement("h2");
nuevoElemento.textContent = "Este es un nuevo encabezado";

let elementoViejo = document.querySelector(“#tituloAntiguo");
document.body.replaceChild(nuevoElemento, elementoViejo);
```

# 9.- Manipulación de nodos en DOM

- **Reemplazar nodos**

  - ✓ **replaceWith():** reemplaza directamente el nodo actual.

    nodoAntiguo.**replaceWith**(nodoNuevo)

  - ✓ **replaceChildren**(): vacía un elemento y agrega nuevos hijos de forma más eficiente.

    lista.**replaceChildren**(…nuevosItems);

# 9.- Manipulación de nodos en DOM

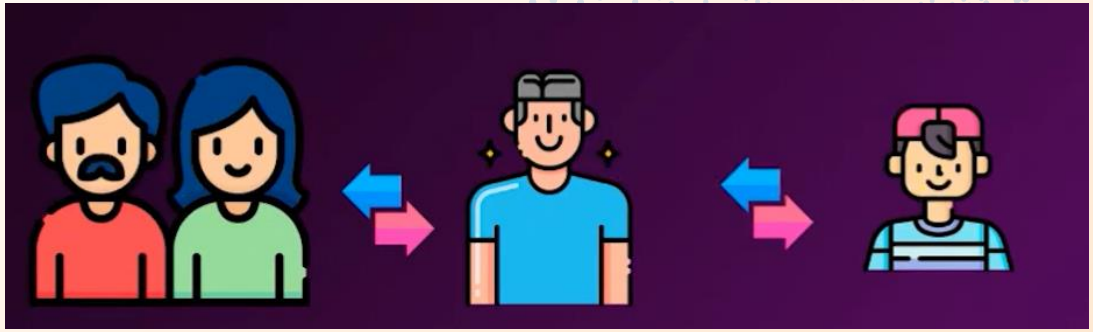
- **Buenas prácticas en modificación estructural**
  - ✓ Evitar modificar el DOM repetidamente dentro de bucles.
  - ✓ Usar **textContent** en lugar de **innerHTML** cuando no se necesita interpretar HTML.
  - ✓ Utilizar **createDocumentFragment()** para inserciones múltiples.

```
const fragmento= document.createDocumentFragment();
// Datos que queremos agregar a la lista
   const frutas = ["Manzana", "Plátano", "Cereza", "Naranja"];

   // Creamos los elementos li y los agregamos al fragmento
   frutas.forEach(fruta => {
    const li = document.createElement("li");
    li.textContent = fruta;
    fragmento.appendChild(li); //añadir en el fragmento, no en el DOM
   });

   // Finalmente agregamos el fragmento al DOM
   const lista = document.getElementById("lista");
   lista.appendfragmento);
```

# 10.- Traversing en el DOM

- El **traversing** es el proceso de recorrer la estructura del DOM para acceder y manipular elementos relacionados.

- Permite acceder a elementos cercanos y realizar modificaciones dinámicas en la estructura de la página



# 10.- Traversing en el DOM

- **Métodos para navegar por los hijos**
  - ✓ **childNodes:** Devuelve una lista de todos los nodos hijos, incluyendo nodos de texto y comentarios.

    *const padre = document.querySelector(“#contenedor");*

    *console.log(padre.childNodes); // NodeList con todos los hijos*
  - ✓ **children:** Devuelve solo los nodos hijos de tipo elemento, excluyendo nodos de texto y comentarios

    *console.log(padre.children); // HTMLCollection con solo elementos*
  - ✓ **firstChild y firstElementChild:** obtienen el primer nodo hijo, mientras que **firtsElementChild** obtiene el primer nodo hijo de tipo element.
  - ✓ **lastChild y lastElementChild:** Similares a los anteriores, pero para el último hijo.

# 10.- Traversing en el DOM

- **Métodos para navegar al nodo padre**
  - ✓ **parentNode:** Devuelve el nodo padre, incluyendo nodos de texto.

    *const hijo = document.getElementById("miHijo");*

    *console.log(hijo.parentNode); // Nodo padre*

  - ✓ **parentElement:** Similar a **parentNode** pero solo devuelve nodos de tipo elemento

    *console.log(padre.children); // HTMLCollection con solo elementos*

# 10.- Traversing en el DOM

- **Métodos para navegar entre nodos hermanos**
  - ✓ **nextSibling:** Accede al siguiente nodo en la lista de nodos, incluyendo nodos de texto.
  - ✓ **nextElementSibling**: Accede al siguiente nodo hermano que es de tipo elemento

```
const elemento = document.querySelector(“#miElemento");
console.log(elemento.nextElementSibling); // Siguiente hermano de tipo elemento
```

# 10.- Traversing en el DOM

- **Métodos para navegar entre nodos hermanos**
  - ✓ **previousSibling:** Accede al nodo anterior en la lista de nodos, incluyendo nodos de texto.
  - ✓ **previousElementSibling**: Accede al hermano anterior de tipo elemento.

```
const elemento = document.querySelector(“#miElemento");
console.log(elemento.previousElementSibling); // Siguiente hermano de tipo elemento
```

# 11.- Uso de jQuery para manipular el DOM

- Aunque JavaScript moderno cubre la mayoría de funciones que antes ofrecía **jQuery**, esta librería sigue siendo muy utilizada en proyectos existentes.
- Muchas librerías y plugins de terceros todavía dependen de *jQuery*, especialmente en entornos empresariales y proyectos que llevan años en producción.
- **jQuery** continúa recibiendo mantenimiento activo, corrigiendo errores, mejorando compatibilidad y garantizando estabilidad para los desarrolladores que aún lo utilizan

.