# UNIDAD 6

## Eventos

# IES MARQUES DE COMARES

**ÍNDICE:**

# IES MARQUES DE COMARES

## 1.- Introducción

En la programación estructurada, las aplicaciones se ejecutan secuencialmente de principio a fin, mientras que en la programación basada en eventos, los programas esperan sin realizar ninguna tarea hasta que se produzca un evento, que es el desencadenante de una tarea asociada a ese evento.

Los eventos de JavaScript permiten la interacción entre las aplicaciones y los usuarios.

Por tanto, un evento es una señal que se genera cuando el usuario actúa con el cliente.

**Ejemplos**:

hacer clic en un botón

Situarse sobre una imagen

Pero, ¿cómo capturar un evento? Con manejadores de eventos como onClick, onLoad, etc.

## 2.- Tipos de eventos

Cada elemento XHTML tiene definida su propia lista de posibles eventos que se le pueden asignar, sabiendo que un mismo tipo de evento puede estar definido para varios elementos XHTML y un mismo elemento XHTML puede tener asociados diferentes eventos.

Para activar un evento, se coloca en la etiqueta en la que ocurrirá el evento.

Un evento comienza con el prefijo **on** seguido de la acción del evento. **ejemplo**, si queremos detectar el evento **click**, el atributo HTML deberá llamarse **onClick**.

## 2.1- Eventos sobre el documento

Son aquellos eventos aplicados sobre todo el documento html.

| Evento | Descripción |
|---|---|
| **onLoad** | Cuando la página página ha terminado de cargarse. |
| **onUnload** | Cuando la página va a cerrarse. |
| **onScroll** | Cuando el usuario hace  scroll sobre la página. |

## 2.2- Eventos sobre la carga de un recurso

Son aquellos eventos aplicados sobre las etiquetas que cargan un archivo externo.

| Evento | Descripción |
|---|---|
| **onLoad** | El recurso ha terminado de cargarse en la página. |
| **onUnload** | El recurso ha sido eliminado de la página. |
| **onAbort** | El recurso ha sido cancelado y no ha terminado su carga. |
| **onError** | El recurso ha dado un error y no ha terminado su carga. |
| **onSelect** | El usuario ha seleccionado un texto de un campo de datos. |

# IES MARQUES DE COMARES

## 2.3- Eventos sobre el focus

Se aplica sobre etiquetas <input>, <textarea>, <select>, <a> o cualquier otra etiqueta que pueda ser seleccionable por el usuario pulsando la tecla TAB

| Evento | Descripción |
|---|---|
| **onBlur** | El elemento ha perdido el foco. |
| **onFocusout** | El elemento ha perdido el foco (se propaga). |
| **onFocus** | El elemento ha ganado el foco. |
| **onFocusin** | El elemento ha ganado el foco (se propaga). |

## 2.4- Eventos sobre el ratón

Cuando el usuario interactúa con el ratón con algún elemento de la página, como podría ser mover el ratón por encima de ellos, hacer clic, etc.

| Evento | Descripción |
|---|---|
| **onClick** | El usuario ha pulsado (y soltado) el elemento. |
| **onDblclick** | El usuario ha hecho doble clic en el elemento. |
| **onMousedown** | El usuario ha pulsado (aún sin haber soltado) el elemento. |
| **onMouseup** | El usuario ha soltado el botón pulsado en un elemento. |
| **onMousemove** | El usuario ha movido el ratón. |
| **onMouseenter** | El usuario ha movido el ratón dentro de un elemento. |
| **onMouseleave** | El usuario ha movido el ratón fuera de un elemento. |
| **onMouseout** | El usuario ha movido el ratón fuera de un elemento (bubbles). |
| **onMouseover** | El usuario ha movido el ratón dentro de un elemento (bubbles). |
| **onWheel** | El usuario ha movido la rueda del ratón. |

## 2.5- Eventos para entrada de datos

Sobre elementos <input> o elementos HTML que son editables por el usuario.

| Evento | Descripción |
|---|---|
| **onBeforeInput** | Un elemento editable justo antes de cambiar. |
| **onInput** | Un elemento editable justo después de que ha cambiado. |

## 2.6- Eventos relacionados con multimedia

Aplicados sobre aquellos elementos que cargan un recurso multimedia.

| Evento | Descripción |
|---|---|
| onEmptied | El audio o video se ha vacíado (recargar elemento). |
| onLoadedMetadata | Se han precargado los metadatos del audio o video (duración, subs...) |
| onLoadedData | Se ha precargado el comienzo del audio o video. |
| onCanPlay | El audio o video se ha precargado lo suficiente para reproducir. |
| onCanPlayThrough | El audio o video se ha precargado completamente. |
| onPlay | El audio o video comienza a reproducirse (tras haber sido pausado). |
| onPlaying | El audio o video comienza a reproducirse. |
| onPause | El audio o video ha sido pausado. |

# IES MARQUES DE COMARES

## Eventos relacionados con multimedia

| Evento | Descripción |
|---|---|
| onTimeUpdate | El audio o video ha avanzado en su reproducción. |
| onEnded | El audio o video ha completado su reproducción. |
| onWaiting | El audio o video está esperando a que el buffer se complete. |
| onDurationChange | El audio o video ha cambiado su duración total (metadatos). |
| onRateChange | El audio o video ha cambiado su velocidad de reproducción. |
| onVolumeChange | El audio o video ha cambiado su volumen de reproducción. |
| onProgress | El audio o video se está descargando. |
| onSeeking | El navegador comenzó a buscar un momento concreto del audio/video. |

# IES MARQUES DE COMARES

## Eventos relacionados con multimedia

| Evento | Descripción |
|---|---|
| onSeeked | El navegador terminó de buscar un momento concreto del audio/video. |
| onResize | El video ha sido redimensionado. |
| onLoadStart | El audio o video ha comenzado a descargarse. |
| onSuspend | La precarga del audio o video ha sido suspendida (ok o pause). |
| onAbort | La precarga del audio o video ha sido abortada o reiniciada. |
| onError | Ha ocurrido un error. |
| onStalled | El navegador intenta precargar el audio o video, pero se ha estancado. |

## Lista completa de eventos

Dada la rapidez de la evolución de los navegadores, es imposible controlar la cantidad de eventos tanto actuales como nuevos que puedan surgir.

Por tanto, para ver en cada momento la lista actualizada de eventos, podéis consultar la Referencia de eventos (evento References) en MDN

https://developer.mozilla.org/es/docs/Web/Events


# IES MARQUES DE COMARES

## 3- Formas de manejar los eventos javascript

- **Evento como si fuese un atributo de Html**
- **Evento como manejador de eventos.**
- **Evento con método addEventListener.**

# IES MARQUES DE COMARES

- **Evento como si fuese un atributo de Html (no es el más recomendable). En este caso podemos llamar a una función predefinida o a cualquier otra que tengamos definida en JS.**

Es la forma más fácil pero menos efectiva, porque realmente no se captura el evento y no podemos consultar nada del mismo.

**Ejemplo:**

<input type="button" value="Enviar" **onclick**="alert('Hola mundo');" />

Se define uno o varios atributos con el nombre o nombres de los eventos.

# IES MARQUES DE COMARES

- **Evento como si fuese un atributo de Html (no es el más recomendable). En este caso podemos llamar a una función predefinida o a cualquier otra que tengamos definida en JS.**

**Ej:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=<device-width>, initial-scale=1.0">
    <title>Document</title>
    <script>
        function Mensaje(mensaje){
            alert(mensaje);
            console.log(event);
        }
    </script>
</head>
<body>
<button onclick="alert('Miguel Angel')">Evento como atributo 1</button>
<button onmouseover="Mensaje(prompt('Introduzca nombre:'))">Evento como atributo 2</button>
</body>
</html>
```

- **Evento como manejador de eventos. Solo hay que crear el evento asociado a un elemento HTML y ejecutarlo.**

Cuando se realizan operaciones complejas, como por ejemplo, la validación de un formulario, se suele agrupar todo el código javascript en una función externa y llamarla desde el elemento XHTML.

**Ejemplo**:

```
function Mostrar() {
  alert(‘Hola Mundo');
}
 <input type="button" value=“Enviar" onclick=“Mostrar()" />
```

# IES MARQUES DE COMARES

- **Evento como manejador de eventos con una sola función asociada al evento**

**Ej:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=<device-width>, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<button id="evento">Evento con manejador </button>
<script>
    function mensaje(){
        alert('Miguel Angel');
        console.log(event);
    }
    //crear evento con manejador y manejarlo
    const $eventoManejador=document.getElementById('evento');
    /*ojo al ejecutarla no se ponen los paréntesis de la función porque si los
    ponemos se ejecutaría inmediatamente, mientras nosotros queremos que se ejecute
    cuando pulsemos el botón correspondiente*/
    $eventoManejador.onclick=mensaje;
</script>
</body>
</html>
```

# IES MARQUES DE COMARES

- **Evento como manejador de eventos con dos funciones asociadas al evento**

**Ej:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=<device-width>, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<button id="evento">Evento con manejador </button>
<script>
    function mensaje1(){
        alert('Miguel Angel');
        }
    function mensaje2(){
        alert('Alejandro');
    }
    //crear evento con manejador y manejarlo
    const $eventoManejador=document.getElementById('evento');
    //siempre que se ejecute un evento lo podemos pasar como event(o e como equivalente)
    $eventoManejador.onclick=function(e){
        mensaje1();
        mensaje2();
        console.log(e);
    }
</script>
</body>
</html>
```


# IES MARQUES DE COMARES

- **Evento con método addEventListener. Sirve para definir múltiples eventos para elementos. De igual forma, podemos eliminar un evento de un elemento con removeEventListener.**

# IES MARQUES DE COMARES

## Evento con escuchador

**Ej:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=<device-width>, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<button id="evento">Evento con listener </button>
<script>
    function mensaje1(){
        alert('Miguel Angel');
        }
    const $eventoListener=document.getElementById("evento");
    //con listener ya no se uliziza la palabra "on" en el evento
    //podemos asociar varias funciones al mismo evento directamente
    $eventoListener.addEventListener("click",mensaje1);
    $eventoListener.addEventListener("click",function(e){
        alert("Alejandro");
        console.log(e);
        console.log(e.type);
        console.log(e.target);
        }
    )
</script>
</body>
</html>
```

# IES MARQUES DE COMARES

## Paso de parámetros a eventos

**Como sabemos, el único parámetro que se le puede pasar a una función manejadora de eventos es el propio evento. Sin embargo, podemos llamar a una función anónima que a su vez llame a otra función que sí lleve parámetros.**

**Ej:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=<device-width>, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<button id="evento">Evento con listener </button>
<script>
    function mensaje(mensaje="Sin nombre"){
        alert(mensaje);
        }
    const $eventoListener=document.getElementById("evento");
    // para pasar parámetros creaos una función y ahí llamamos a la función que sí lleva
    // parámetros porque la función manejadora de eventos es la arrow que no lleva parámetros
    $eventoListener.addEventListener("click",(e)=>{mensaje("Miguel Angel");console.log(e)});
</script>
</body>
</html>
```



## Obtener información del evento

Una vez obtenido el evento, podemos acceder a toda la información del mismo mediante sus propiedades.

**Ejemplo**:

```
function manejadorEventos2(e) {
    var coordenadaX = e.clientX;
    var coordenadaY = e.clientY;
    alert(`Posición:  (${coordenadaX},${coordenadaY})`);
    }
document.onclick = manejadorEventos2;
```

# IES MARQUES DE COMARES

## 5.- Propiedades del objeto event

**(fuente: https://developer.mozilla.org/es/docs/Web/API/Event)**

**event.altKey :** Devuelve un valor indicando si la tecla <alt> fue pulsada durante el evento.

**event.bubbles:** Devuelve un valor que indica si el evento se propaga o no hacia arriba a través del DOM

**event.button:** Devuelve el botón pulsado del ratón.

**event.cancelBubble:** Devuelve un valor que indica si la propagación hacia arriba fue cancelada o no.

**event.cancelable:** Devuelve un valor que indica si el evento se puede cancelar.

**event.charCode:** Devuelve el valor Unicode de una tecla de carácter que fue apretada como parte de un evento keypress.

**event.clientX:** Devuelve la posición horizontal del evento.

**event.clientY:** Devuelve la posición vertical del evento.

**event.ctrlKey:** Devuelve un valor que indica si la tecla <Ctrl> fue apretada durante el evento.

**event.currentTarget:** Devuelve una referencia al objetivo actual registrado para el evento**.**

**event.detail:** Devuelve detalles sobre el evento, dependiendo del tipo de evento.

**event.eventPhase:** Utilizado para indicar qué fase del flujo del evento es actualmente en proceso de evaluación.

**event.isChar:** Devuelve un valor que indica si el evento produce o no una tecla de carácter.

**event.keyCode:** Devuelve el valor Unicode de una tecla que no es caracter en un evento keypress o cualquier tecla en cualquier otro tipo de evento de teclado.



# 6.- Métodos del objeto event

**(fuente: https://developer.mozilla.org/es/docs/Web/API/Event)**

**ent.initEvent:** Inicia el valor de un evento que se ha creado vía la interfaz DocumentEvent.
**event.initKeyEvent:** Inicia un evento del teclado.
**event.initMouseEvent:** Inicia un evento del ratón una vez que se ha creado.
**event.initUIEvent:** Inicia un evento de la interfaz de usuario (UI) una vez que se ha creado.
**event.preventBubble:** Previene la expansión del evento. Este método es desaconsejado en favor del estándar stopPropagation.
**event.preventCapture:** Este método es desaconsejado en favor del estándar stopPropagation.
**event.preventDefault:** Cancela el evento (si éste es anulable).
**event.stopPropagation:** Para la propagación de los eventos más allá en el DOM.

# IES MARQUES DE COMARES

## Propiedades y métodos objeto

**event.layerX:** Devuelve la coordenada horizontal del evento relativo a la capa actual.
**event.layerY:** Devuelve la coordenada vertical del evento relativo a la capa actual.
**event.metaKey:** Devuelve un valor booleano indicando si la meta tecla fue presionada durante un evento.
**event.pageX:** Devuelve la coordenada horizontal del evento, relativo al documento completo.
**event.pageY:** Devuelve la coordenada vertical del evento, relativo al documento completo.
**event.relatedTarget:** Identifica un objetivo secundario para el evento.
**event.screenX:** Devuelve la coordenada horizontal del evento en la pantalla.
**event.screenY:** Devuelve la coordenada vertical del evento en la pantalla.
**event.shiftKey:** Devuelve un valor booleano indicando si la tecla <shift> fue presionada cuando el evento fue disparado.
**event.target:** Devuelve una referencia al objetivo en la cual el evento fue originalmente enviado.
**event.timeStamp:** Devuelve el momento de creación del evento.
**event.type:** Devuelve el nombre del evento (distingue mayúsculas y minúsculas).
**event.view:** El atributo vista identifica la AbstractView del cual el evento fue generado.
**event.which:** Devuelve el valor Unicode de la tecla en un evento del teclado, sin importar el tipo de tecla que se presionó.

# IES MARQUES DE COMARES

Para entender bien este método, primero veamos lo que significa el movimiento o flujo de eventos (captura y burbuja).

Supongamos el siguiente código html:

```
<body>
   <div>
      <button>Enviar</button>
   </div>
</body>
```

Pues bien, al hacer click sobre el botón Enviar, además es como si hiciésemos también click sobre los elementos que lo contienen (padres) del DOM, es decir, es como si hubiésemos hecho también click sobre además sobre el elemento body y sobre el elemento div.

Por tanto, sí sólo hay una función asignada a una escucha para el botón no hay mayor problema, pero si además hay otra escucha para el body y otra para el div, ¿cuál es el orden en que ejecutarán esas funciones?

## Anexo: Bubbling de Eventos (burbujeo)

El Bubbling hace referencia al desencadenamiento de eventos en elementos anidados desde el más interno hasta el más externo al igual que una burbuja.

Veámoslo con un ejemplo de página en la que tenemos tres cuadrados incluido uno interno dentro de otro y este, a su vez, dentro de otro más externo.

# IES MARQUES DE COMARES

## Ejemplo página bubbling

**Archivo css**

```
body,html{
    height: 100%;
    margin: 0;}
.fondo{
  background-color: rgb(72, 182, 233) ;
  height: 100%;
  width: 100%; }
.cuadroexterno{
  background-color: blue ;
  width: 400px;
  height: 400px; }
.cuadromedio{
  background-color: green;
  width: 200px;
  height: 200px; }
.cuadrointerno{
  background-color: red ;
  width: 100px;
  height: 100px;
.fondo,.cuadroexterno,.cuadromedio,.cuadrointerno{
    align-items: center;
    display: flex;
    justify-content: center;
    }
```

# IES MARQUES DE COMARES

## Ejemplo página bubbling

### Archivo js

```
function mostrarfondo(){
        alert("fondo");
        }
function mostrarexterno(){
        alert("cuadro externo");
        }
function mostrarmedio(){
        alert("cuadro medio");
        }
function mostrarinterno(){
        alert("cuadro interno");
        }
```

# IES MARQUES DE COMARES

## Ejemplo página bubbling

**Archivo html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Ejemplo burbujeo</title>
    <link href="ejemploburbujeo.css" rel="stylesheet" type="text/css" />
    <script src="ejemplobubbling.js"></script>
</head>
<body>
    <div class="fondo" onclick="mostrarfondo();">
        <div id="cuadroexterno" class="cuadroexterno" onclick="mostrarexterno();">
            <div id="cuadromedio" class="cuadromedio" onclick="mostrarmedio();">
                <div id="cuadrointerno" class="cuadrointerno" onclick="mostrarinterno()">
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

## 4.- Obtener información del evento

JavaScript permite obtener información del evento mediante el objteto event.

Para Internet explorer el objeto evento forma parte del objeto window, mientras que el resto de navegadores lo consideran como el único argumento de la función manejadora del evento, ya que todos ellos crean de forma automática un argumento que se pasa a la función manejadora del evento, por lo que no es necesario incluirlo en la llamada a la función.

Por tanto, para usar dicho argumento basta con darle un nombre.

**Ejemplo que funciona en todos los navegadores**:

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
}
```

# IES MARQUES DE COMARES

**Archivo html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Ejemplo burbujeo</title>
    <link href="ejemploburbujeo.css" rel="stylesheet" type="text/css" />
    <script src="ejemplobubbling.js"></script>
</head>
<body onload="inicializar()">
    <div class="fondo" onclick="mostrarfondo();">
        <div id="cuadroexterno" class="cuadroexterno" onclick="mostrarexterno();">
            <div id="cuadromedio" class="cuadromedio" onclick="mostrarmedio();">
                <div id="cuadrointerno" class="cuadrointerno" onclick="mostrarinterno()">
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

# IES MARQUES DE COMARES

## Veamos el mismo ejemplo del burbujeo obteniendo información del evento producido

```
function inicializar(){
        var cuadroint=document.getElementById("cuadrointerno");
        var cuadromed=document.getElementById("cuadromedio");
        var cuadroext=document.getElementById("cuadroexterno");
        var fondo=document.getElementById("fondo");
        cuadroint.addEventListener("click",mostrarinterno,false);
        cuadromed.addEventListener("click",mostrarmedio,false);
        cuadroext.addEventListener("click",mostrarexterno,false);
        fondo.addEventListener("click",mostrarfondo,false);
        }
function mostrarfondo(e){
        console.log(e);
        alert(`Se ha producido el evento ${e.type} por el elemento ${e.target.id}`);
        }
function mostrarexterno(e){
        alert(`Se ha producido el evento ${e.type} por el elemento ${e.target.id}`);
        }
function mostrarmedio(e){
        alert(`Se ha producido el evento ${e.type} por el elemento ${e.target.id}`);
        }
function mostrarinterno(e){
        alert(`Se ha producido el evento ${e.type} por el elemento ${e.target.id}`);
        }
```


# IES MARQUES DE COMARES

Una forma más correcta sería utilizar el método addEventListener para recorger el evento deseado sobre un elemento del documento.

Para el mismo ejemplo anterior, pero realizado con addEventListener, el archivo css sería el mismo, pero los archivos html y js podrían ser, por ejemplo, los mostrados en las dos páginas siguientes.

# Método addEventListener()

El flujo de eventos significa que cuando se hace clic sobre un elemento, el evento se propaga en puede propagar en dos sentidos:

- uno que es la captura, es decir, el evento comienza en el nivel superior del documento y recorre los elementos de padres a hijo
- otro la burbuja, es decir, el orden inverso, ascendiendo de hijos a padres.

Así, el orden por defecto de ejecución de las supuestas funciones sería body-div-button, pero podemos alterar dicho sentido para ejecutarlas en el sentido button-div-body.

Los dos sentidos anteriores del flujo son los que van a poder configurarse en el parámetro booleano del método addEventListener (false: burbujeo y true: captura)

# Método addEventListener()

El método addEventListener nos permite establecer un listener sobre un elemento de la página, para capturar el evento producido al interactuar con dicho elemento.

**Sintaxis:**

elemento.addEventListener('evento',función,boolean); donde:

elemento: cualquier elemento de una página al que accedemos por su id, por su etiqueta o las propiedades de otro nodo.

evento: es el suceso ocurrido sobre el elemento

Función: cualquier función que queramos que se ejecute cuando ocurra el evento.

booleano: valor que define el orden del flujo de eventos, de forma que si es true se propaga de padres a hijos y si es false se propaga de hijo a padre.

# Método addEventListener()

**Ejemplo**:
Queremos validar un formulario antes de ser enviado al servidor mediante un botón 'Enviar'.

En el formulario supondría tener algo así:

```
<button type="submit" id="Enviar">Enviar </button>
```

y en el script, podríamos capturar el evento sobre el botón de la forma:

```
document.getElementById('Enviar').addEventListener('click',validar,false);
```

Nota:
Aquí lo llamativo es que si Javascript no puede ser ejecutado en la máquina cliente, de todas formas el formulario será enviado.

## Método removeEventListener()

Pero, ¿Qué ocurre si necesito escuchar un evento sólo una vez? Lo podemos controlar con el método removeEventListener con sintaxis:

```
elemento.removeEventListener('evento',funciónremovida,booleano);
```

# IES MARQUES DE COMARES

## Ejemplo página bubbling con addEventListener

**Archivo html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Ejemlo burbujeo</title>
    <link href="ejemploburbujeo2.css" rel="stylesheet" type="text/css" />
    <script src="ejemplobubbling2.js"></script>
</head>
<body onload="inicializar()"">
    <div id="fondo" class="fondo">
        <div id="cuadroexterno" class="cuadroexterno" >
            <div id="cuadromedio" class="cuadromedio" >
                <div id="cuadrointerno" class="cuadrointerno" >
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

# IES MARQUES DE COMARES

## Ejemplo página bubbling con addEventListener

**Archivo js**

```
function inicializar(){
        var cuadroint=document.getElementById("cuadrointerno");
        var cuadromed=document.getElementById("cuadromedio");
        var cuadroext=document.getElementById("cuadroexterno");
        var fondo=document.getElementById("fondo");
        cuadroint.addEventListener("click",mostrarinterno,true);
        cuadromed.addEventListener("click",mostrarmedio,true);
        cuadroext.addEventListener("click",mostrarexterno,true);
        fondo.addEventListener("click",mostrarfondo,true)
        }
function mostrarfondo(){
        alert("fondo");
        }
function mostrarexterno(){
        alert("cuadro externo");
        }
function mostrarmedio(){
        alert("cuadro medio");
        }
function mostrarinterno(){
        alert("cuadro interno");
        }
```

Problema: solo tengo cuatro div, pero imaginemos que tenemos 1000, tengo que generar 1000 variables y 1000 addEventListener.

# IES MARQUES DE COMARES

## Ejemplo página bubbling con addEventListener

**Solución al problema del ejemplo anterior (creando una clase que englobe a todos los div)**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Ejemlo burbujeo</title>
    <link href="ejemploburbujeo2.css" rel="stylesheet" type="text/css" />
</head>
<body>
<section class="flujo-eventos">
    <div id="fondo" class="fondo">
        <div id="cuadroexterno" class="cuadroexterno" >
            <div id="cuadromedio" class="cuadromedio" >
                <div id="cuadrointerno" class="cuadrointerno" >
                </div>
            </div>
        </div>
    </div>
</section>
<script src="ejemplobubbling3.js"></script>
</body>
</html>
```

# IES MARQUES DE COMARES

## Ejemplo página bubbling con addEventListener

```
const $divs=document.querySelectorAll(".flujo-eventos div");


$divs.forEach(div=>{
        div.addEventListener("click",flujoEventos);
        });

function flujoEventos(e){
        // en este caso this hace referencia al div que lo llama
        console.log(`elemento ${this.className}, propagado de ${e.target.className}`)
;
```

# IES MARQUES DE COMARES

## ¿Cómo detener la propagación de eventos?

Podemos detener la propagación de eventos simplemente usando el método stopPropagation().

Ej:

```
const $divs=document.querySelectorAll(".flujo-eventos div");


$divs.forEach(div=>{
        div.addEventListener("click",flujoEventos);
        });

function flujoEventos(e){
        // en este caso this hace referencia al div que lo llama
        console.log(`elemento ${this.className}, propagado de ${e.target.className}`)
;
        e.stopPropagation();
}
```

# 7.- Delegación de eventos

Lo más interesante sería poder definir un evento en un elemento padre y luego delegar su funcionamiento a todos sus hijos, por la optimización de la memoria

Por eso, lo mejor es definir un evento en el elemento document, con lo cual, se podría detectar en cualquiera de los hijos del documento.

Ventaja: Ya no tendríamos que asignar ese evento a cada uno de los elementos que haya dentro del documento (imaginemos que queremos comprobar el evento click en los 30 elementos de un formulario, solo lo definiríamos una vez en el document y no 30 veces) ni tendríamos que comprobar el stopPropagation, ya que el evento está definido en el elemento más externo y no existe burbujeo hasta el más externo (siempre que esté definido en modo burbuja y no captura).



## Delegación de eventos

**En definitiva, lo más elegante y efectivo es definir de modo global en el document, todos los eventos que vayamos a tratar y a partir de ahí detectaríamos el nodo que activaría cada evento y desencadenaríamos la programación correspondiente.**

# IES MARQUES DE COMARES

## Delegación de eventos

**Veamos la aplicación del evento click aplicado al documento en nuestro ejemplo del burbujeo.**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Ejemlo burbujeo</title>
    <link href="ejemploburbujeo2.css" rel="stylesheet" type="text/css" />
</head>
<body>
<section class="flujo-eventos">
    <div id="fondo" class="fondo">
        <div id="cuadroexterno" class="cuadroexterno" >
            <div id="cuadromedio" class="cuadromedio" >
                <div id="cuadrointerno" class="cuadrointerno" >
                </div>
            </div>
        </div>
    </div>
</section>
<script src="ejemplobubbling3.js"></script>
</body>
</html>
```

# IES MARQUES DE COMARES

## Delegación de eventos

**Veamos la aplicación del evento click aplicado al documento en nuestro ejemplo del burbujeo. Si ejecutamos, veremos que nos muestra el elemento en el que hacemos click sin burbujeo, al estar definido el Listener en el documento y no en cada uno de los elementos.**

```
document.addEventListener("click",e =>{
        console.log("click en ",e.target);
// con el condicional buscamos el elemento al que deseamos aplicar el evento
// en nuestro caso por ejemplo, si hacemos click en algún div, ejecutamos la función fl
ujoEventos
        if (e.target.matches(".flujo-eventos div")){
                flujoEventos(e);
        }
});

function flujoEventos(e){
        // en este caso this hace referencia al div que lo llama
        console.log(`elemento ${this}, propagado de ${e.target.className}`);
        // ya no necesitamos stopPropagation como antes, por lo que podemos quitarlo
        //e.stopPropagation();
        }
```