# IES MARQUES DE COMARES

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Junta de Andalucía
Consejería de Educación y Deporte

Unión Europea-
NextGenerationEU.

# 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, podemos colocarlo en la etiqueta en la que ocurrirá el evento. En este caso, 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. |

## 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 vaciado (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. |

## 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. |


# IES MARQUES DE COMARES

## 3- Formas de manejar los eventos javascript

- Eventos como atributos HTML
- Manejadores como funciones externas
- Manejadores semánticos
- addEventListener()



# 3.1.- Eventos como atributos HTML

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.

## 3.2.- Manejadores con funciones externas

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()" />
```

## 3.3.- Manejadores de eventos semánticos

Se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos.

**Ejemplo**:

```
// Función externa
function mostrar() {
        alert('Hola Mundo');
        }
/* Asignar la función externa al elemento. Observar que se llama a la función
   sin paréntesis.*/
document.getElementById("Enviar").onclick = mostrar;
// Elemento XHTML
 <input id="Enviar" type="button" value="Enviar" />
```

# Limitaciones de los manejadores de eventos semánticos

- En primer lugar, solo podemos asignar una función a un evento manejado como evento semántico. Es decir, si le asignamos dos funciones, se queda con la última (es como si la sobreescribiera).

Ej: // Función externa
function mostrar1() { alert('Hola Mundo1'); }
function mostrar2() { alert('Hola Mundo2'); }
document.getElementById("Enviar").onclick = mostrar1;
document.getElementById("Enviar").onclick = mostrar2;
En este ejemplo, ya no tendría ningún sentido mostrar1, ya que habría sido reemplazada por mostrar2

- En segundo lugar, no se pueden pasar parámetros a la función manejadora de eventos semánticos, o mejor dicho, solo se puede pasar el parámetro predefinido event.

Ejs: document.getElementById("Enviar").onclick=function(){alert("Hola mundo1");}
document.getElementById("Enviar").onclick=function(e){
console.log("Hola mundo2");
console.log(e); // e es lo mismo que event
console.log(event);
}

## 3.4.- 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 u objeto]); 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.
- Objeto: objeto con la propiedad predefinidas capture u once principalmente que pueden tomar los valores true o false:

  {capture:false, once:true} significa que se ejecutará en modo burbuja, pero solo una vez, es decir, cuando se ha ejecutado onclick una vez, es como si se deshabilitaran los onclick en todos los elementos.

# IES MARQUES DE COMARES

## Método addEventListener()

El método addEventListener nos permite asignar varias funciones al mismo manejador de eventos, no como en el anterior que se sobreescribía.

Ej:

```
document.getElementById("Enviar").addEventListener("click",function(){
                                                    alert("Hola mundo1")});
document.getElementById("Enviar").addEventListener("click",function(e){
            alert("Hola mundo2");
            console.log(e); // e es event
            console.log(event);
            });
```

## 3.4.1.- Bubling de eventos(burbujeo)

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 sobr además sobre el elemento body y sobre el elemento div.

Por tanto, sí sólo hay un evento asociado al botón, no hay mayor problema, pero si además el mismo evento está asociado para el div y para el body, ¿cuál es el orden en que ejecutarán las funciones asociadas a esos eventos?


# IES MARQUES DE COMARES

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.

# 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 onclick="mostrarbody();";>
    <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

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 js

```
function mostrarExterno(){
        alert("cuadro externo");
        }
function mostrarMedio(){
        alert("cuadro medio");
        }
function mostrarInterno(){
        alert("cuadro interno");
        }
function mostrarFondo(){
        alert("fondo");
        }
function mostrarBody(){
        alert("body");
        }
```

# IES MARQUES DE COMARES

## Ejemplo página bubbling

Archivo css

```css
body,html{
    height: 100%;
    margin: 0;
}
.fondo,.cuadroexterno,.cuadromedio,.cuadrointerno{
    align-items: center;
    /*la caja flexible o FlexBox con display: flex. Funciona ajustando los tamaños y la disposición de los elementos que se encuentran dentro de un contenedor o caja, de tal manera que se adapten siempre al espacio disponible.*/
    display: flex;
    /*La propiedad justify-content define cómo el navegador distribuye el espacio entre y alrededor de los items de la caja flex*/
    justify-content: center;
}
.fondo{
   background-color: rgb(72, 182, 233) ;
   height: 100%;
   position: relative;
   width: 100%;
}
 .cuadroexterno{
   background-color: blue ;
   width: 600px;
   height: 600px;
}
 .cuadromedio{
    background-color: green;
    width: 300px;
    height: 300px;
}
 .cuadrointerno{
     background-color: red ;
     width: 150px;
     height: 150px;
}
```


# 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.

# 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,false);
        cuadromed.addEventListener("click",mostrarMedio,false);
        cuadroext.addEventListener("click",mostrarExterno,false);
        fondo.addEventListener("click",mostrarFondo,false);
}
function mostrarExterno(){
        alert("cuadro externo");
        }
function mostrarMedio(){
        alert("cuadro medio");
        }
function mostrarInterno(){
        alert("cuadro interno");
        }
function mostrarFondo(){
        alert("fondo");
        }
```

# IES MARQUES DE COMARES

**Ejemplo Método addEventListener()**:
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);
```

Lo importante 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]);

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
- funciónremovida: cualquier función (declarada o expresada, nunca anónima) 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.

Ej:

```
const quitandoClick = (e)=>{
        alert("quitando el evento Click");
        console.log(e.type,e.target);
        document.getElementById("Enviar").removeEventListener("click",quitandoClick);
        }

document.getElementById("Enviar").addEventListener("click",quitandoClick);
```

## 3.4.2.- Deshabilitar eventos Bubling

Vamos a ver cómo detener el bubling, para que solo se ejecute una vez el onclick, y se desactive en todos los elementos del bubling

**Sintaxis:**

elemento.addEventListener('evento',función,[boolean u objeto]);

En este caso vamos a utilizar como parámetro objeto con dos variables predefinidas capture y once, para elegir el método de propagación y desactivar los eventos respectivamente.

**Ejemplo**: todos funcionarán en modo burbuja, pero una vez hecho click en cualquiera de ellos se deshabilita el evento onclick

```
cuadroint.addEventListener("click",mostrarInterno,{capture:false,once:true});
cuadromed.addEventListener("click",mostrarMedio,{capture:false,once:true});
cuadroext.addEventListener("click",mostrarExterno,{capture:false,once:true});
fondo.addEventListener("click",mostrarFondo,{capture:false,once:true});
```

# IES MARQUES DE COMARES

## 3.4.3.- Detener propagación eventos Bubling

Se usa el método del evento stopPropagation.

**Ejemplo**:

```
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 mostrarExterno(e){
        alert("cuadro externo");
           e.stopPropagation();
        }
function mostrarMedio(e){
        alert("cuadro medio");
            e.stopPropagation();
        }
function mostrarInterno(e){
        alert("cuadro interno");
            e.stopPropagation();
        }
function mostrarFondo(e){
        alert("fondo");
            e.stopPropagation();
        }
```

## 3.4.4.- Delegación de eventos

Es la forma idónea de trabajar con eventos.

Consiste en aplicar un único evento al elementos superior (document) y posteriormente ir detectando qué elemento ( o mejor dicho detectamos con el método matches qué selector) ha provocado ese evento

De esta forma, evitamos tener varios elementos a los que se le aplica el mismo evento y además evitamos la propagación de eventos.

# IES MARQUES DE COMARES

## Ejemplo anterior con Delegación de eventos

```
function inicializar(){
    document.addEventListener("click",(e)=>{
        console.log(e.target);
        if (e.target.matches("#cuadrointerno")){
            mostrarInterno();
        }
        if (e.target.matches("#cuadromedio")){
            mostrarMedio();
        }
        if (e.target.matches(".cuadroexterno")){
            mostrarExterno();
        }
        if (e.target.matches(".fondo")){
            mostrarFondo();
        }
    })
}
function mostrarExterno(e){
    alert("cuadro externo");
    }
function mostrarMedio(e){
    alert("cuadro medio");
    }
function mostrarInterno(e){
    alert("cuadro interno");
    }
function mostrarFondo(e){
    alert("fondo");
    }
```

## 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;
}
```



# Obtener información del evento

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

**Ejemplos**:

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
  var informaciónevento = "Tipo de evento: " + evento.type + "<br>" +
                          "keyCode: " + evento.keyCode + "<br>" +
                          "charCode: " + evento.charCode + "<br>" +
                           “tecla: “+ String.fromCharCode(evento.charCode);
}
document.onclick=manejadorEventos;

function manejadorEventos2(elEvento) {
  var evento = elEvento || window.event;
  var coordenadaX = evento.clientX;
  var coordenadaY = evento.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.