Índice: Tema 3.5

acer ejercici de 1 a 23

Índice: Tema 3.5

El modelo de eventos de JavaScript

- ✓ **JavaScript**, además de ser un lenguaje **basado en objetos**, es también un lenguaje de programación **conducido por eventos**.
- ✓ Los scripts de JavaScript que aparecen en las páginas web se pueden ejecutar de dos formas:
  - Durante la interpretación del código de la página, a medida que se van encontrando las instrucciones ejecutables (incluyendo posiblemente invocaciones de funciones).
  - Al producirse determinados eventos para los que se han previsto acciones concretas (scripts de código asociado).
- ✓ En el primer método el código se ubica dentro de la etiqueta script sin estar asociado a ningún evento (Ejercicio 1).
- ✓ Vamos a analizar aquí con más detenimiento este segundo método de ejecución de código: **ocurrencia de eventos**.

Eventos y manejadores de eventos

- ✓ Un evento es una señal que se dispara al producirse algún cambio.
  - Los eventos son provocados la mayor parte de las veces por las acciones del usuario (a medida que maneja el ratón o el teclado).
  - Aunque también pueden ser provocados por otros tipos de cambios en el entorno, como los que ocurren durante el procesamiento del documento recibido (fin de la carga del documento, abandono de la página, etc…).
- ✓ Cuando tiene lugar un evento se ejecuta automáticamente el script asociado a su **manejador** (cada evento tiene asociado un manejador).
- ✓ Por defecto los manejadores de eventos no tienen scripts asociados, de forma que no se hace nada cuando se producen.
- ✓ Somos nosotros los que tenemos que asociar scripts a los manejadores de los eventos que queramos controlar.

Tipos de eventos (1)

- **Eventos de ratón:** Soportados por todos los elementos.

`click`	Pulsar y soltar el ratón.
`dblclick`	Pinchar dos veces seguidas con el ratón.
`mousedown`	Pulsar un botón del ratón y no soltarlo.
`mousemove`	Mover el ratón.
`mouseout`	El ratón sale del elemento.
`mouseover`	El ratón entra en el elemento.
`mouseup`	Soltar el botón del ratón.

- **Eventos de teclado:** Soportados por todos los elementos de formulario y <body>.

`keydown`	Pulsar una tecla y no soltarla.
`keypress`	Pulsar una tecla.
`keyup`	Soltar una tecla pulsada.

Tipos de eventos (2)

- **Eventos de carga:** <body>.

`abort`	Interrumpir la carga de una imagen antes de ser cargada.
`error`	Produce un error durante la carga de una imagen.
`load`	Terminar la carga del documento.
`unload`	Abandonar el documento.

- **Eventos de ventana:** <body>.

`move`	Desplazar la ventana.
`resize`	Cambiar el tamaño de la ventana.
`scroll`	Hacer scroll en el documento.

- **Eventos de enfoque:** <body>, <button>, <input>, <label>, <select> y <textarea>.

`blur`	Deseleccionar el elemento.
`focus`	Seleccionar el elemento.

Tipos de eventos (3)

Eventos de formulario

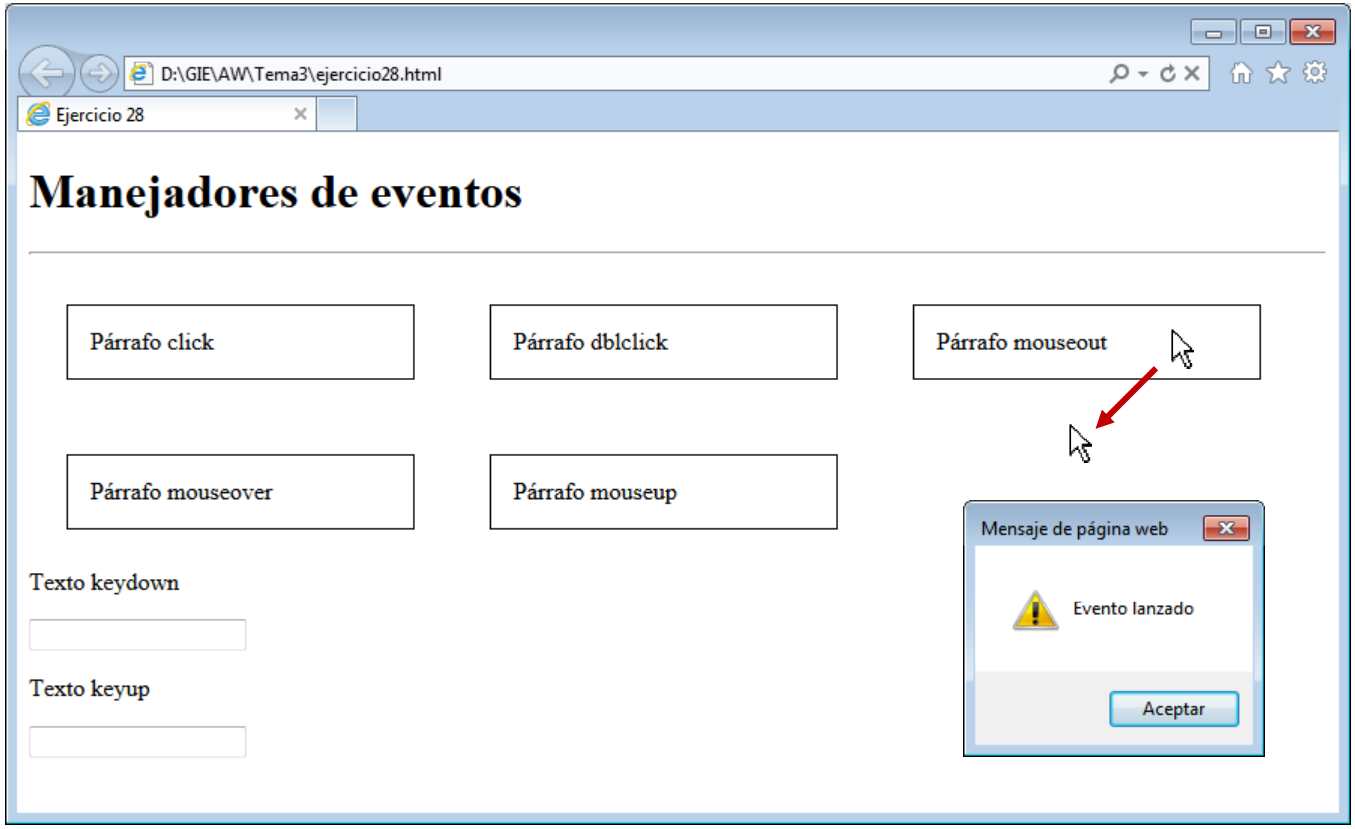
`change`	Deseleccionar un elemento que se ha modificado (<input>, <select> y <textarea>).
`reset`	Inicializar el formulario (<form>).
`select`	Seleccionar un texto (<input> y <textarea>).
`submit`	Enviar el formulario (<form>).

- Los manejadores de dichos eventos se llaman igual, pero añadiendo el prefijo "`on`" delante. Por ejemplo:

```
onclick
onblur
onkeypress
```

➢ Tipos de eventos (4)

✓ Probar el lanzamiento de diferentes eventos asociando al correspondiente manejador del evento una función que muestre el mensaje “Evento lanzado”.



Propagación de eventos (1)

- ✓ Existen determinados eventos que solo son relevantes para unos elementos determinados:
  - Por ejemplo, el evento **`submit`** se produce sobre un **`<form>`**, pero no afecta a otras clases de objetos como **`<textarea>`**, etc…
- ✓ Sin embargo, hay otros eventos que son relevantes a varios elementos. Entonces, ¿qué ocurre si asociamos scripts a los mismos manejadores de evento de varios elementos anidados?
  - Por ejemplo, si se asocian distintos manejadores al evento **`click`** del cuerpo y de un párrafo dentro del cuerpo. ¿Qué ocurre al hacer click sobre el párrafo?
- ✓ Respuesta: Por defecto, los eventos se van propagando hacia arriba por la jerarquía de contenidos y todos los objetos antecesores tienen la posibilidad de responder al evento (siempre que sea relevante) hasta llegar al objeto raíz.

Propagación de eventos (2)

- Analizar cómo se propaga el evento **`click`** asociando una alerta diciendo dónde se ha pulsado a los manejadores **`onclick`** del cuerpo, los dos párrafos y los elementos negrita y cursiva de la siguiente página web.

Índice: Tema 3.5

Manejadores de eventos

- ✓ Cada evento se dispara cuando se da la condición asociada y se produce sobre un determinado objeto.
- ✓ Los componentes de la página web descritos en su archivo como elementos HTML tienen asociado un objeto JavaScript durante la visualización de la página.
- ✓ Para que el objeto de un elemento HTML reaccione a un determinado evento que se produzca sobre él, hay que asociar al correspondiente manejador el script adecuado. Esto se puede hacer de diferentes formas:
  - **Como atributos**: Código JavaScript dentro de un atributo del propio elemento HTML. El código también se puede ubicar en una función externa y proporcionar su nombre como atributo.
  - **Manejadores semánticos**: Asignados mediante el DOM sin necesidad de modificar el código HTML de la página.

Manejadores como atributos (1)

- La forma más sencilla de incluir un manejador de evento es mediante un atributo HTML con el mismo nombre del manejador del evento que se quiere procesar.
- El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento.

```
<input type="button" value="Botón" onclick="alert('Hiciste click')" />
```

- Cuando el código de la función manejadora es más complejo, es aconsejable agrupar todo el código JavaScript en una función externa:

```
<script>
     function myfunc() {
        alert('Hiciste click');
     }
</script>

<input type="button" value="Botón" onclick="myfunc()" />
```

Manejadores como atributos (2)

- En los eventos de JavaScript, se puede utilizar la palabra reservada **this** para referirse al elemento HTML sobre el que se está ejecutando el evento:

```
<div id="micaja" style="width:200px; height:200px; background-color:black"
     onmouseover = "document.getElementById('micaja').style.backgroundColor='red'"
     onmouseout  = "document.getElementById('micaja').style.backgroundColor='black'">
</div>
```

- Mediante el uso de **this**, este código puede reescribirse sin necesidad de utilizar las funciones del DOM para acceder al objeto correspondiente:

```
<div id="micaja" style="width:90px; height:90px; background-color:black"
     onmouseover = "this.style.backgroundColor = 'red'"
     onmouseout  = "this.style.backgroundColor = 'black'">
</div>
```

Manejadores como atributos (3)

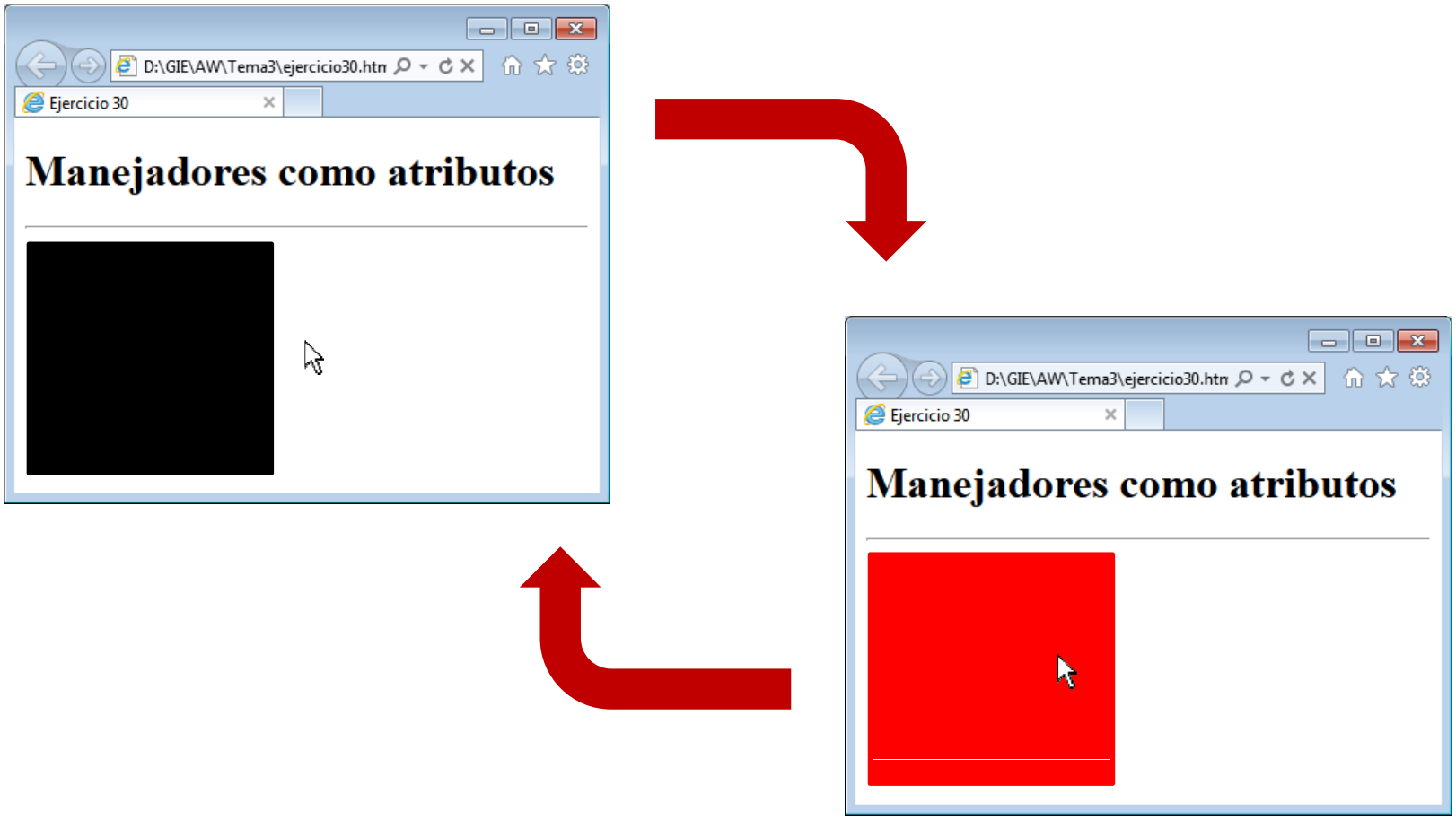
- En las funciones externas no es posible utilizar la variable **this** directamente, por lo que hay que pasarla como parámetro a la función manejadora:

```
function funCaja(objmicaja) {
    switch(objmicaja.style.backgroundColor) {
        case 'black':
        case 'rgb(0,0,0)':
        case '#000000':
            objmicaja.style.backgroundColor = 'red';
            break;
        case (('red')||('rgb(255,0,0)')||('#FF0000')):
            objmicaja.style.backgroundColor = 'black';
            break;
        default:
            alert("El color inicial no es no rojo ni negro");
    }
}

<div id="micaja" style="width:200px; height:200px; background-color:black"
     onmouseover = "funCaja(this)"
     onmouseout  = "funCaja(this)">
</div>
```

Manejadores como atributos (4)

- ✓ Probar el código del ejemplo anterior:



Manejadores semánticos (1)

- ✓ Al igual que se recomienda separar los contenidos (XHTML) de la presentación (CSS), también se recomienda separar los contenidos de la programación (JS).
- ✓ Mezclar JavaScript y XHTML complica excesivamente el código fuente de la página, dificulta su mantenimiento y reduce la semántica del documento final producido.
- ✓ Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript sin usar atributos: los **manejadores semánticos**.
- ✓ Esta técnica consiste en asignar las funciones externas, que contienen el código a ejecutar, a los manejadores de eventos correspondientes mediante las propiedades DOM de los elementos XHTML. Para ello:
  - Se crea una función de JavaScript encargada de manejar el evento correspondiente.
  - Se asigna un identificador único al elemento HTML en cuestión mediante el atributo id.
  - Se asigna la función a un evento concreto del elemento HTML mediante DOM.

Manejadores semánticos (2)

- ✓ Debe tenerse en cuenta que para asignar los manejadores de eventos mediante las funciones DOM, es necesario que la página se haya cargado por completo para que el DOM se haya creado antes de acceder a dichas funciones.
- ✓ Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento **`load`** del objeto **`window`**. Así, el ejemplo de la transparencia 13, quedaría:

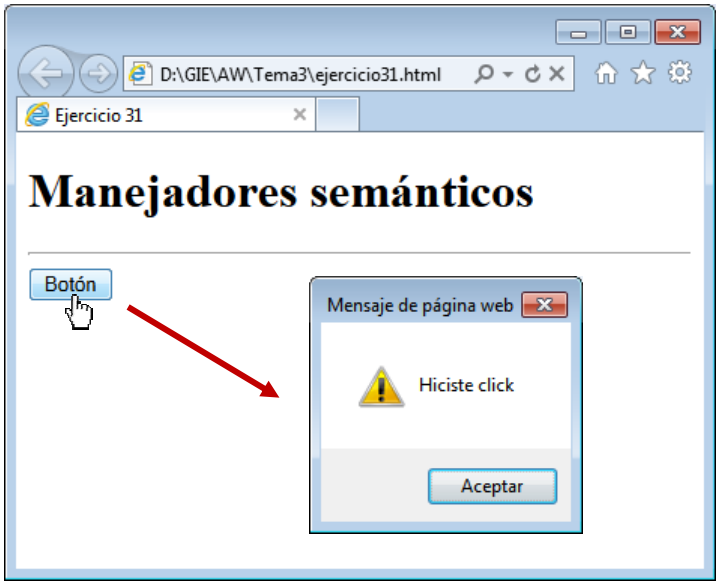
```
<script>
    function myfunc() {
        alert("Hiciste click");
    }
    window.onload = function() {
        document.getElementById("miboton").onclick = myfunc;
    }
</script>
<input id="miboton" type="button" value="Botón" />
```

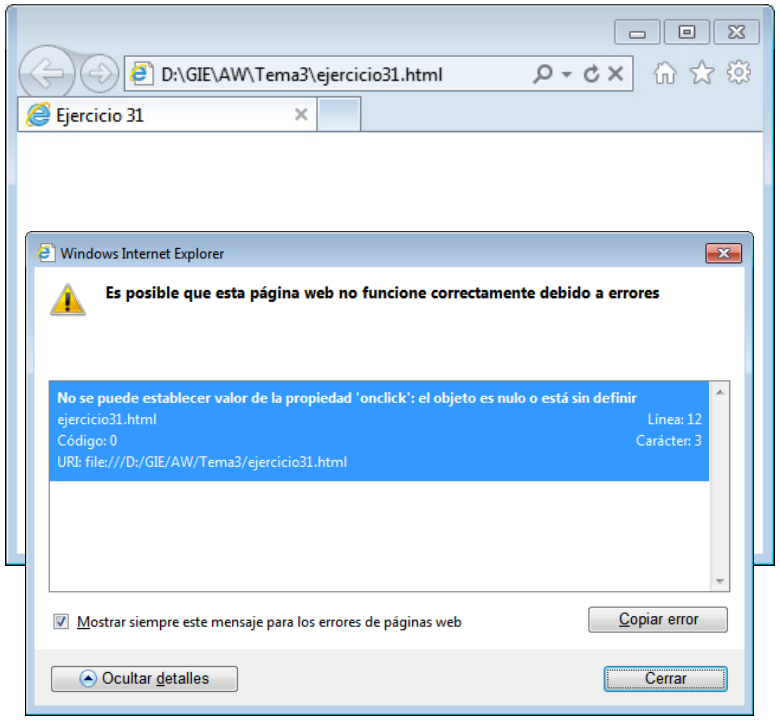
Manejadores semánticos (3)

- ✓ Nótense un par de detalles en el ejemplo anterior:
  - La asignación al evento `load` del objeto `window` de todas las instrucciones que se quieren ejecutar tras la carga de la página se hace mediante una función anónima.
  - La asignación de la función manejadora (`myfunc`) al manejador `onclick` del elemento en cuestión se hace sin incluir los paréntesis:
    - Nótese que es una asignación de la función `myfunc` al manejador y no una llamada a la misma.
    - Si se incluyesen los paréntesis en dicha asignación no estaríamos asignado dicha función al manejador `onclick`, sino el resultado de ejecutar la función `myfunc`.
- ✓ Veremos más adelante como se pueden pasar argumentos a la función manejadora (en caso de que los necesite) utilizando manejadores semánticos (objeto `event`).

Manejadores semánticos (4)

- ✓ Probar el código del ejemplo de la transparencia 18 y ver que funciona correctamente. Adicionalmente, ver que ocurre si se trata de acceder a DOM antes de cargar la página, o si se asigna `myfunc` utilizando `myfunc()`.





Manejadores semánticos (5)

- ✓ Para terminar con los manejadores semánticos, veamos cómo se pueden asignar **varias funciones manejadoras a un mismo manejador de eventos**.
- ✓ La especificación DOM define los métodos `addEventListener()` y `removeEventListener()` para asociar y desasociar manejadores de eventos.
- ✓ Los métodos requieren tres parámetros:
  - El nombre entre comillas del evento que se quiere manejar.
  - Una referencia a la función encargada de procesar el evento.
  - Si el tercer parámetro es `true`, el manejador se emplea en el modelo de propagación de eventos descendente (*capturing*), si es `false` en el ascendente (*bubbling*).
- ✓ En versiones anteriores a IE9 se emplean los métodos `attachEvent()` y `detachEvent()`, en donde no existe el tercer parámetro y el primero es el manejador del evento en cuestión.

Manejadores semánticos (6)

- Ahora, el código del la transparencia 18 se puede reescribir para asignar dos funciones manejadoras al evento **`click`** del botón como:

```
<script>
    function myfunc1() {
        alert('Hiciste click - manejador 1');
    }
    function myfunc2() {
        alert('Hiciste click - manejador 2');
    }
    window.onload = function() {
        var objmiboton=document.getElementById("miboton")
        objmiboton.addEventListener("click",myfunc1,false);
        objmiboton.addEventListener("click",myfunc2,false);
        // objmiboton.removeEventListener("click",myfunc2,false); para desasociar
    }
</script>
<input id="miboton" type="button" value="Botón" />
```

Manejadores semánticos (7)

- ✓ Modificar el código del ejemplo anterior para que funcione en todos los navegadores:

Índice: Tema 3.5

El objeto event

- En ocasiones, cuando se produce un evento, no es suficiente con asignarle al manejador correspondiente la función responsable de procesar ese evento.
- En estos casos, la función que procesa el evento necesita información relativa al evento producido:
  - La tecla que se ha pulsado, la posición del ratón (coordenadas en pantalla) o el elemento que ha producido el evento.
- El objeto `event` es el mecanismo definido por los navegadores para proporcionar toda esa información.
- Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignadas al evento.

Acceso al objeto event de un elemento

- Internet Explorer permite el acceso al objeto **event** a través del objeto **window**. Así, dentro de la función encargada de procesar el evento se puede acceder al objeto **event** mediante:

```
function funProcesadora() {
    var objevent = window.event;
}
```

- En el estándar DOM el objeto **event** es el único parámetro que se debe pasar a las funciones encargadas de procesar los eventos. Así, el acceso al objeto **event** se puede hacer de dos formas diferentes:

```
function funProcesadora() {
    var objevent = arguments[0];
}
function myfunc(objevent) {
}
```

Propiedades del objeto event en IE (1)

- La siguiente tabla recoge las propiedades definidas para el objeto `event` en los navegadores de la familia Internet Explorer.
- Todas las propiedades salvo `repeat` son de lectura/escritura y por tanto, su valor se puede leer y/o establecer.

Propiedad/ Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla ALT y `false` en otro caso
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: `0` – Ningún botón pulsado 1 – Se ha pulsado el botón izquierdo 2 – Se ha pulsado el botón derecho 3 – Se pulsan a la vez el botón izquierdo y el derecho 4 – Se ha pulsado el botón central 5 – Se pulsan a la vez el botón izquierdo y el central 6 – Se pulsan a la vez el botón derecho y el central 7 – Se pulsan a la vez los 3 botones
`cancelBubble`	Boolean	Si se establece un valor `true`, se detiene el flujo de eventos de tipo *bubbling*

Propiedades del objeto event en IE (2)

clientX	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
clientY	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
ctrlKey	Boolean	Devuelve `true` si se ha pulsado la tecla CTRL y `false` en otro caso
fromElement	Element	El elemento del que sale el ratón (para ciertos eventos de ratón)
keyCode	Número entero	En el evento `keypress`, indica el carácter de la tecla pulsada. En los eventos `keydown` y `keyup` indica el código numérico de la tecla pulsada
offsetX	Número entero	Coordenada X de la posición del ratón respecto del elemento que origina el evento
offsetY	Número entero	Coordenada Y de la posición del ratón respecto del elemento que origina el evento
repeat	Boolean	Devuelve `true` si se está produciendo el evento `keydown` de forma continuada y `false` en otro caso
returnValue	Boolean	Se emplea para cancelar la acción predefinida del evento
screenX	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
screenY	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa

Propiedades del objeto event en IE (3)

shiftKey	Boolean	Devuelve `true` si se ha pulsado la tecla SHIFT y `false` en otro caso
srcElement	Element	El elemento que origina el evento
toElement	Element	El elemento al que entra el ratón (para ciertos eventos de ratón)
type	Cadena de texto	El nombre del evento
x	Número entero	Coordenada X de la posición del ratón respecto del elemento padre del elemento que origina el evento
y	Número entero	Coordenada Y de la posición del ratón respecto del elemento padre del elemento que origina el evento

Propiedades del objeto event en DOM (1)

- La siguiente tabla recoge las propiedades definidas para el objeto **`event`** en los navegadores que siguen los estándares DOM.
- Al contrario de lo que sucede con Internet Explorer, la mayoría de propiedades del objeto **`event`** de DOM son de sólo lectura. En concreto, solamente las siguientes propiedades son de lectura y escritura: **`altKey`**, **`button`** y **`keyCode`**.

Propiedad/Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla ALT y `false` en otro caso
`bubbles`	Boolean	Indica si el evento pertenece al flujo de eventos de *bubbling*
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: 0 – Ningún botón pulsado 1 – Se ha pulsado el botón izquierdo 2 – Se ha pulsado el botón derecho 3 – Se pulsan a la vez el botón izquierdo y el derecho 4 – Se ha pulsado el botón central 5 – Se pulsan a la vez el botón izquierdo y el central 6 – Se pulsan a la vez el botón derecho y el central 7 – Se pulsan a la vez los 3 botones

Propiedades del objeto event en DOM (2)

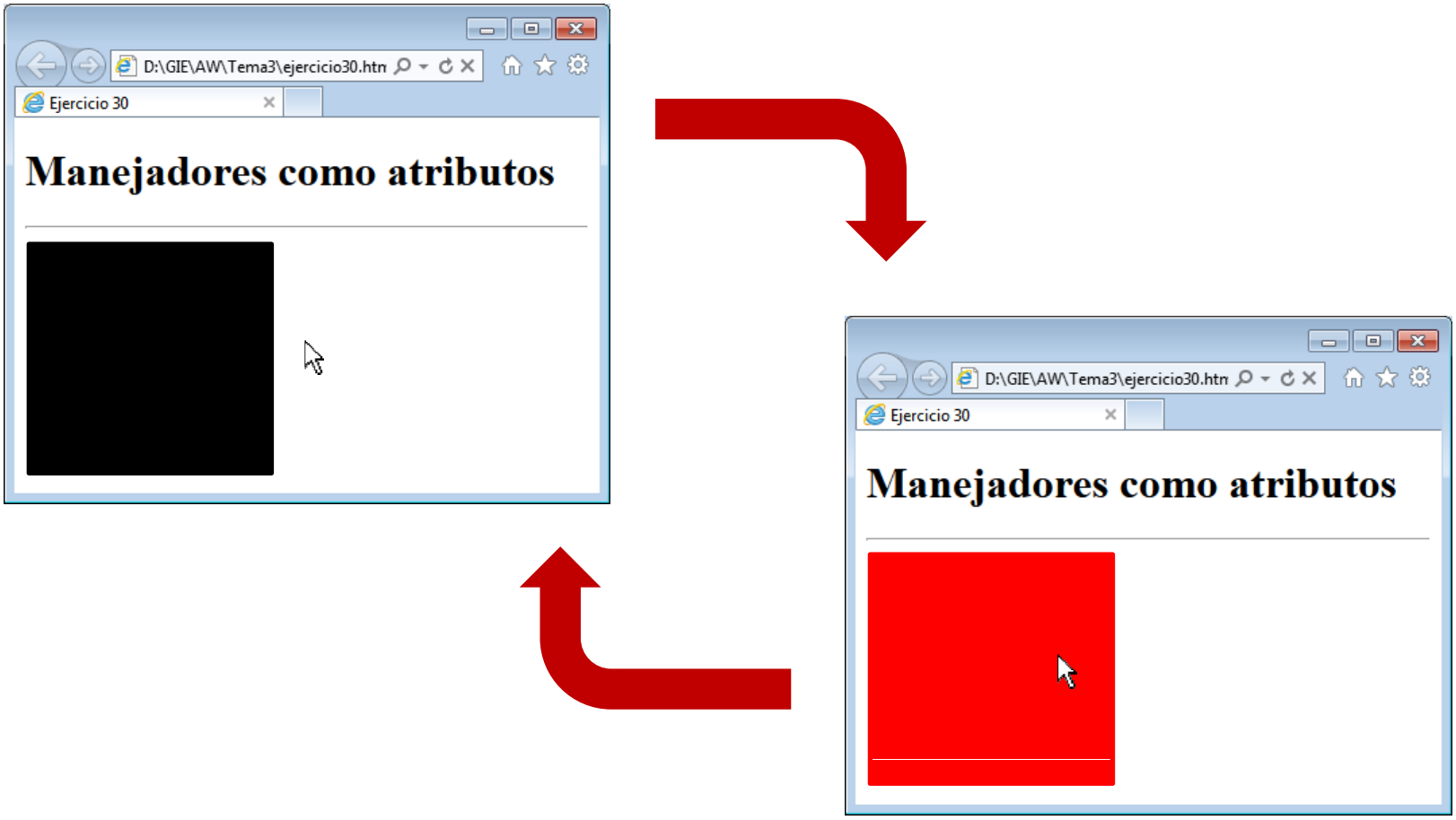
cancelable	Boolean	Indica si el evento se puede cancelar
cancelBubble	Boolean	Indica si se ha detenido el flujo de eventos de tipo *bubbling*
charCode	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
clientX	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
clientY	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
ctrlKey	Boolean	Devuelve `true` si se ha pulsado la tecla CTRL y `false` en otro caso
currentTarget	Element	El elemento que es el objetivo del evento
detail	Número entero	El número de veces que se han pulsado los botones del ratón
eventPhase	Número entero	La fase a la que pertenece el evento: `0` – Fase capturing `1` – En el elemento destino `2` – Fase bubbling
isChar	Boolean	Indica si la tecla pulsada corresponde a un carácter
keyCode	Número entero	Indica el código numérico de la tecla pulsada

Propiedades del objeto event en DOM (3)

metaKey	Número entero	Devuelve `true` si se ha pulsado la tecla META y `false` en otro caso
pageX	Número entero	Coordenada X de la posición del ratón respecto de la página
pageY	Número entero	Coordenada Y de la posición del ratón respecto de la página
preventDefault()	Función	Se emplea para cancelar la acción predefinida del evento
relatedTarget	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
screenX	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
screenY	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
shiftKey	Boolean	Devuelve `true` si se ha pulsado la tecla SHIFT y `false` en otro caso
stopPropagation()	Función	Se emplea para detener el flujo de eventos de tipo *bubbling*
target	Element	El elemento que origina el evento
timeStamp	Número	La fecha y hora en la que se ha producido el evento
type	Cadena de texto	El nombre del evento

➢ Accediendo al objeto event

✓ Repetir el Ejercicio 30 utilizando manejadores semánticos (IE y DOM):



Índice: Tema 3.5

Tipos de eventos

- La lista completa de eventos que se pueden generar en un navegador se puede dividir en cuatro grandes grupos:
  - **Eventos de ratón**: se originan cuando el usuario emplea el ratón para realizar algunas acciones.
  - **Eventos de teclado**: se originan cuando el usuario pulsa sobre cualquier tecla de su teclado.
  - **Eventos HTML**: se originan cuando se producen cambios en la ventana del navegador o cuando se producen ciertas interacciones entre el cliente y el servidor.
  - **Eventos DOM**: se originan cuando se produce un cambio en la estructura DOM de la página. También se denominan "eventos de mutación".

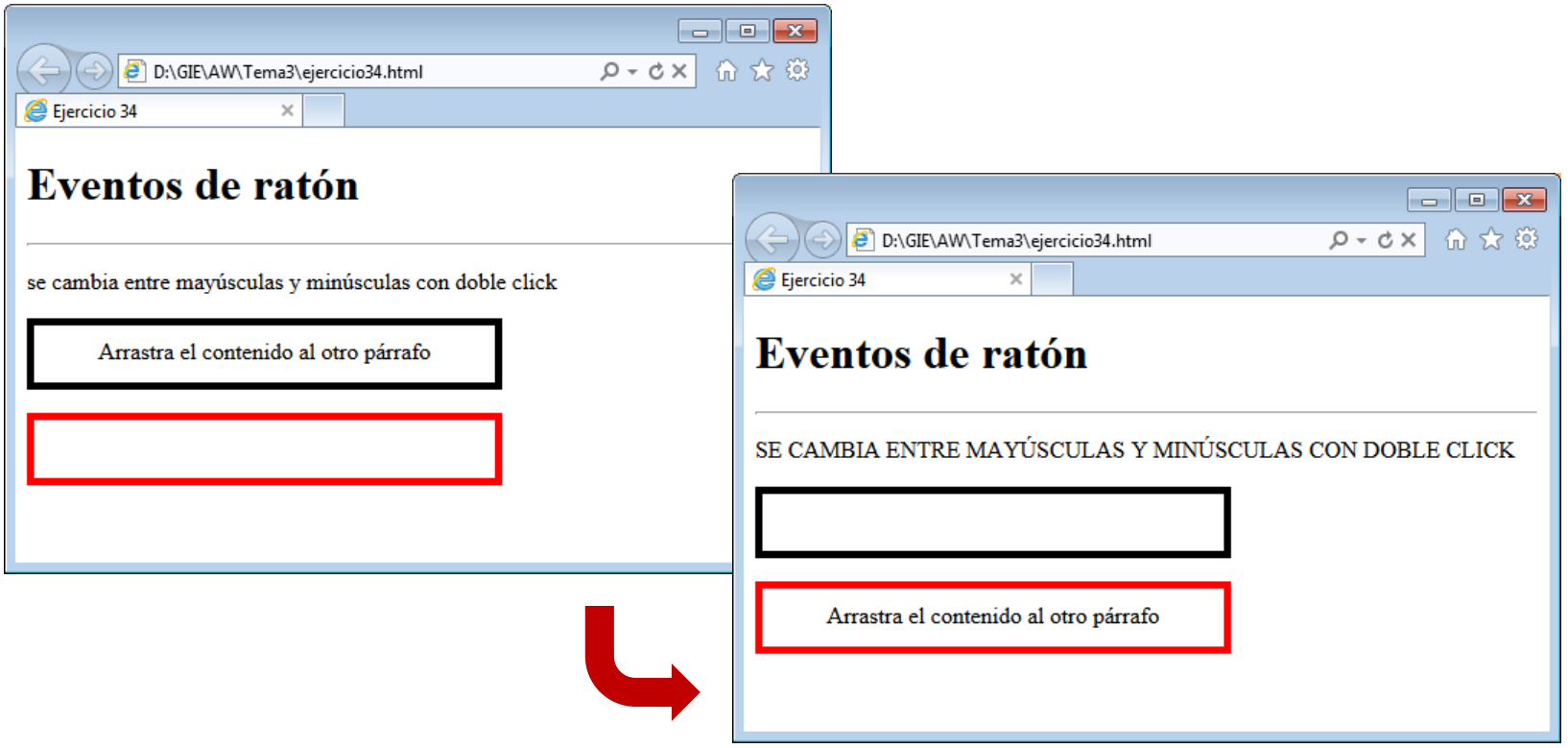
Eventos de ratón (1)

- ✓ Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente: **`mousedown`**, **`mouseup`**, **`click`**.

- ✓ El objeto **`event`** contiene las siguientes propiedades para los eventos de ratón:
  - Las coordenadas del ratón (todas las coordenadas diferentes relativas a los distintos elementos)
  - La propiedad **`type`**.
  - La propiedad **`srcElement`** (IE) o **`target`** (DOM).
  - Las propiedades **`shiftKey`**, **`ctrlKey`**, **`altKey`** y **`metaKey`** (sólo DOM).
  - La propiedad **`button`** (sólo en los eventos **`mousedown`**, **`mousemove`**, **`mouseout`**, **`mouseover`** y **`mouseup`**).

Eventos de ratón (2)

- ✓ Programar las funciones a asociar a los manejadores de eventos correspondientes de los tres párrafos siguientes para alternar entre mayúsculas y minúsculas con un doble click en el 1º y arrastrar el contenido del 2º y 3º y viceversa.



Eventos de teclado (1)

- ✓ Cuando se pulsa una tecla correspondiente a un carácter alfanumérico, se produce la siguiente secuencia de eventos: **`keydown`**, **`keypress`**, **`keyup`**.
- ✓ Cuando se pulsa otro tipo de tecla, se produce la siguiente secuencia de eventos: **`keydown`**, **`keyup`**.
- ✓ El objeto **`event`** contiene las siguientes propiedades para los eventos de teclado:
  - La propiedad **`keyCode`**.
  - La propiedad **`charCode`** (sólo DOM).
  - La propiedad **`srcElement`** (IE) o **`target`** (DOM).
  - Las propiedades **`shiftKey`**, **`ctrlKey`**, **`altKey`** y **`metaKey`** (sólo DOM).

Eventos de teclado (2)

- ✓ Programar las funciones a asociar a los manejadores de eventos correspondientes del cuerpo del documento para cambiar el color de fondo del mismo en función de la pulsación de diferentes teclas.

Eventos de teclado

Al pulsar las teclas R, G y B se cambia de color de fondo, cualquier otra tecla invierte el color. Si se suelta la tecla en cuestión la página vuelve a su estado original

Eventos de teclado

Al pulsar las teclas R, G y B se cambia de color de fondo, cualquier otra tecla invierte el color. Si se suelta la tecla en cuestión la página vuelve a su estado original

Eventos de teclado

Al pulsar las teclas R, G y B se cambia de color de fondo, cualquier otra tecla invierte el color. Si se suelta la tecla en cuestión la página vuelve a su estado original