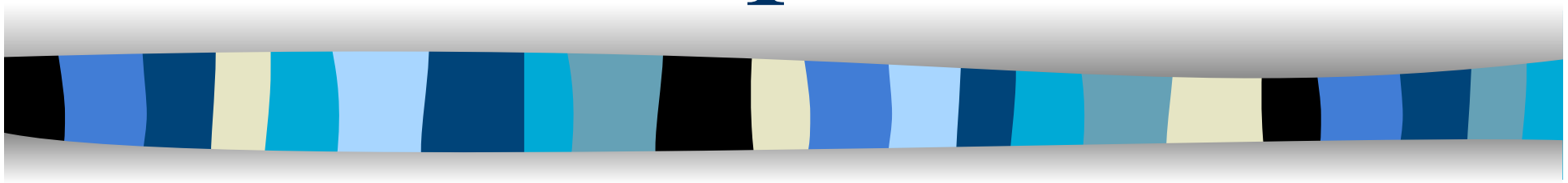
JavaScript – 4



Eventi

Gestore di evento inline

```
<p onmouseover="this.style.background='white';
                this.style.color='black'">
   Lorem ipsum dolor sit amet, consectetur … </p>

<p onmouseover="this.style.background='white';
                this.style.color='black'"
   onmouseout="this.style.removeProperty('color');
               this.style.removeProperty('background')">
   Lorem ipsum dolor sit amet, consectetur ….
</p>
```

ripasso

Registrando funzioni

```
<script type="text/javascript">
        function handleMouseOver(elem) {
            elem.style.background='white';
            elem.style.color='black';
        }

        function handleMouseOut(elem) {
            elem.style.removeProperty('color');
            elem.style.removeProperty('background');
        }
    </script>
```

ripasso

… in HTML

```
<body>
 <p onmouseover="handleMouseOver(this)"
      onmouseout="handleMouseOut(this)">
         Lorem ipsum dolor sit …
 </p>
 <p onmouseover="handleMouseOver(this)"
      onmouseout="handleMouseOut(this)">
    I need your help, Luke. She needs your help. I am …
      </p>
</body>
```

ripasso

Registrazione di eventi – 1

```
<p>Qui scriviamo qualcosa...</p>
<p id="block2">Qui continuiamo...</p>
```

```
<script type="text/javascript">
 var pElems = document.getElementsByTagName("p");
  for (var i = 0; i < pElems.length; i++) {
        pElems[i].onmouseover = handleMouseEvent;
        pElems[i].onmouseout = handleMouseEvent;
   }
```

Registrando gli eventi in questo modo, la funzione associata all'evento ha come parametro l'oggetto **Event**

Registrazione di eventi – 2

```
function handleMouseEvent(e) {
      if (e.type == "mouseover") {
          e.target.style.background='white';
          e.target.style.color='black';
      } else {
          e.target.style.removeProperty('color');
          e.target.style.removeProperty('background');
                }
}
</script>
```

Note – 1

- Nel codice precedente il nome della funzione è stato usato per registrarlo come un *listener* di un evento
- Un errore comune consiste nel mettere le parentesi dopo il nome
- Questo avrebbe l'effetto di chiamare la funzione quando lo script è eseguito e non quando l'evento è scatenato
  - All'evento è associato il risultato della valutazione della funzione

Note – 2

- Il parametro *e* della funzione sarà settato ad un oggetto **Event** creato dal browser al momento in cui è scatenato l'evento.
- L'oggetto **Event** consente maggiore flessibilità rispetto ad associare codice Javascript quale valore di un attributo onXXXX di un elemento HTML
- Nell'esempio è utilizzata la proprietà *target* di **Event** per individuare l'elemento HTML su cui poi agire per modificare lo stile.

Alcune proprietà di Event

Name	Description	Returns
`type`	The name of the event (e.g., `mouseover`).	`string`
`target`	The element at which the event is targeted.	`HTMLElement`
`currentTarget`	The element whose event listeners are currently being invoked.	`HTMLElement`
`eventPhase`	The phase in the event life cycle.	`number`
`bubbles`	Returns `true` if the event will bubble through the document, `false` otherwise.	`boolean`
`cancelable`	Returns `true` if the event has a default action that can be cancelled, `false` otherwise.	`boolean`
`timeStamp`	The time at which the event was created, or `0` if the time isn't available.	`string`
`stopPropagation()`	Halts the flow of the event through the element tree after the event listeners for the current element have been triggered.	`void`
`stopImmediatePropagation()`	Immediately halts the flow of the event through the element tree; untriggered event listeners for the current element will be ignored.	`void`
`preventDefault()`	Prevents the browser from performing the default action associated with the event.	`void`
`defaultPrevented`	Returns `true` if `preventDefault()` has been called.	`boolean`

Ancora su registrazione di eventi – 1

- L'oggetto **HTMLElement** implementa dei metodi per la gestione degli eventi
  - Restituito da un'invocazoine dei metodi getElementById, getElementByTagName,…
- Il metodo *addEventListener* è utilizzato per associare una funzione ad un evento
- Il metodo *removeEventListener* rimuove l'associazione tra evento e funzione.

Esempio

```
..//codice come slide precedenti
<button id="pressme">Rimuovi handler</button>

<script type="text/javascript">
 var pElems = document.getElementsByTagName("p");
 for (var i = 0; i < pElems.length; i++) {
        pElems[i].addEventListener("mouseover", handleMouseOver);
        pElems[i].addEventListener("mouseout", handleMouseOut);
}

document.getElementById("pressme").onclick =
   function() {
     var elem= document.getElementById("block2");
     elem.removeEventListener("mouseout", handleMouseOut);
   }
```

```
function handleMouseOver(e) {
    e.target.style.background='white';
    e.target.style.color='black';
}

function handleMouseOut(e) {
    e.target.style.removeProperty('color');
    e.target.style.removeProperty('background');
}
```

Ancora su registrazione di eventi -2

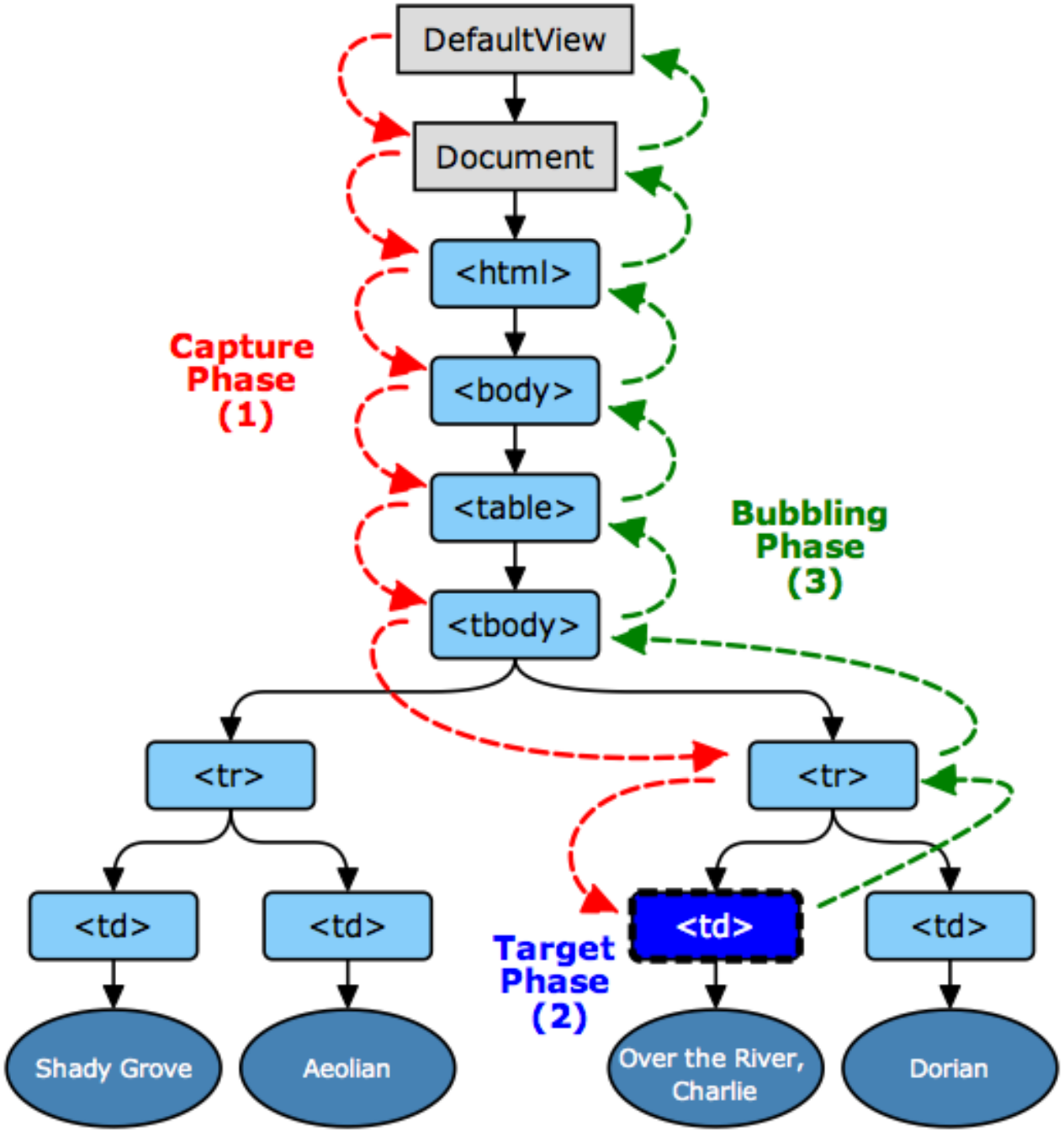
- Nello script precedente quando si preme il bottone, l'invocazione del metodo *removeEventListener* rompe l'associazione tra evento e funzione
  - Si possono utilizzare tecniche differenti contemporaneamente.
- Il vantaggio nell'utilizzare il metodo *addEventListener* risiede nella possibilità di utilizzare caratteristiche avanzate della gestione degli eventi

Perché usare *addEventListener* ?

- Permette di aggiungere diversi handler per singolo evento
- Fornisce un controllo migliore di cosa succede quando il listener viene attivato
  - capture vs bubbling (dettagli in seguito)
- Funziona con qualunque elemento DOM, non solo con gli elementi HTML
  - document, window, XMLHttpRequest

Flusso degli eventi

- Un evento ha tre fasi nel suo ciclo di vita:
  - Capture (cattura)
  - Target (obiettivo)
  - Bubbling (salita a bolla)
- In ogni fase possono essere eseguiti gli handler associati ad un evento


DefaultView
Document
<html>
<body>
<table>
<tbody>
<tr>
<tr>
<td>
<td>
<td>
<td>
Shady Grove
Aeolian
Over the River, Charlie
Dorian
Capture Phase (1)
Target Phase (2)
Bubbling Phase (3)

La fase di cattura

- Quando è *scatenato* un evento evt, il browser individua l'elemento a cui si riferisce l'evento (detto *target*)
- Il browser individua tutti gli *elementi* tra DefaultView (window), passando per document, per l'elemento html arrivando fino al target e per ognuno di essi
  - controlla se esiste un gestore (di evento) che ha chiesto di essere notificato di eventi di tipo evt dei suoi discendenti
  - scatena questi gestori prima di scatenare quelli del target

Eventi e fase di cattura

```
var textblock = document.getElementById('b1');

textblock.addEventListener("mouseover",
                           handleDescendantEvent, true);
```

- Nella registrazione abbiamo aggiunto un terzo parametro settato a true
- In questo modo abbiamo indicato al browser di catturare tutti gli eventi di tipo mouseover scatenati in uno qualsiasi dei discendenti dell'elemento con id settato al valore b1

Esempio (con un *problema*)

```
<p id="p1"
style="border:solid red 1px; padding: 5px; width: 40%;">
 Inizio del paragrafo
<span id="s1" style="border:dashed blue  1px;">
testo in span</span>
fine del paragrafo ..... clicca prima sullo span poi qui
</p>
```

```
paragrafo = document.getElementById('p1');
span = document.getElementById('s1');
paragrafo .addEventListener("click", salutaP, true);
span.addEventListener("click", salutaS);
```

Funzioni javascript – 1

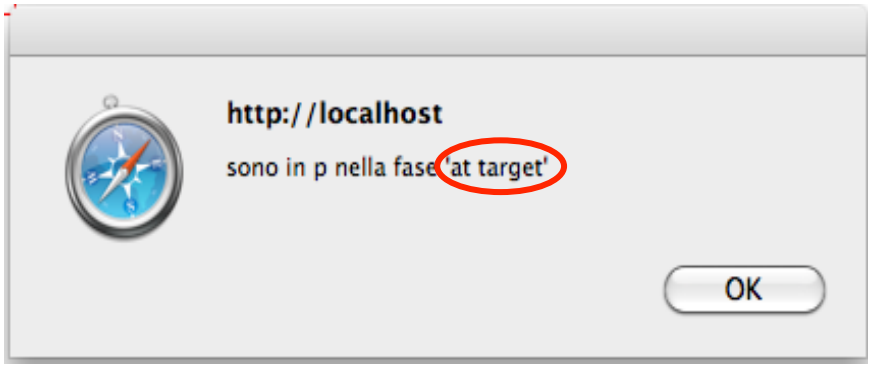
```
function salutaP(e){
 alert('sono in p nella fase \" +
getPhaseName(e.eventPhase) + '\");
}

function salutaS(e){
 alert('sono in span nella fase \" +
getPhaseName(e.eventPhase) + '\");
}
```

Funzioni javascript – 2

```
function getPhaseName(numPhase) {
 switch(numPhase)
 {
   case 1:
     return 'capture';  break;

   case 2:
     return 'target';  break;

   case 3:
     return 'bubbling'; break;

   default:
     return 'unknown';
  }
}
```

Problema

- Cliccando nel paragrafo al di fuori dello span otteniamo il seguente messaggio



Con l'istruzione addEventListener("click", salutaP, true); abbiamo comunque registrato sul paragrafo il gestore salutaP associato all'evento click (non solo rispetto alla fase di **cattura**

Per eseguire salutaP **solo** durante la fase di **cattura** dobbiamo modificare il codice della funzione salutaP

Correzione

```
function salutaP(e){
if (e.eventPhase == Event.CAPTURING_PHASE)
 alert('sono in p nella fase \" +
        getPhaseName(e.eventPhase) + '\");
}
```

Valori che può assumere eventPhase

- CAPTURING_PHASE
- AT_TARGET
- BUBBLING_PHASE

target e currentTarget

- Sono entrambe proprietà di Event
- target **Nell'esempio precedente il tag span**
  - Si riferisce all'elemento che è l'obiettivo dell'evento
  - L'elemento dal quale l'evento ha avuto origine
- currentTarget **Nell'esempio precedente il tag p**
  - Si riferisce all'elemento i cui gestori di eventi sono attualmente invocati
  - L'elemento su cui l'attuale gestore di eventi è stato registrato

Fase di target

- Eseguita dopo la fase di capture
- È la fase più semplice da gestire
- Il browser scatena ogni listener per il tipo di evento che è stato aggiunto all'elemento target

Fase di bubbling

- Eseguita dopo la fase di target
- Il browser risale l'abero fino a DefaultView e, per ogni elemento che incontra, controlla se ci sono eventi (del tipo scatenato) registrati con il terzo parametro settato a *false*.

```
var textblock = document.getElementById('b1');

textblock.addEventListener("mouseover",
                           handleDescendantEvent, false);
```

```
textblock.addEventListener("mouseover",
                           handleDescendantEvent);
```

Nota – 1

- Ricordarsi aggiungere al gestore dell'evento il test per controllare che siamo nella fase *giusta*

```
if (e.eventPhase == Event.BUBBLING_PHASE)
```

- Non tutti gli eventi supportano il bubbling. Per verificare se è supportato è sufficiente controllare la proprietà bubbles
  - true → l'evento supporta il bubbling
  - false → l'evento non supporta il bubbling

Nota – 2

- Quando si registra un evento con il metodo addEventListener il listener dell'elemento riceverà sempre una notifica dagli elementi discendenti
- Possiamo scegliere se invocare il listener prima della fase di target (capture phase – terzo parametro true) o dopo (bubbling phase – terzo parametro false)
  - Se non indichiamo il terzo parametro esso è settato di default a false

Interrompere il flusso di un evento

- Si possono usare due metodi di Event
- stopPropagation()
  - Assicurerà che tutti i gestori di eventi registrati per l'elemento corrente saranno invocati
- stopImmediatePropagation()
  - Ignora qualsiasi gestore di evento che non sia stato già scatenato

Dettagli negli esempi

Eventi cancellabili – 1

- Alcuni eventi hanno una *azione di default* che viene eseguita quando si scatena l'evento.
- Nel tag A, l'evento *click*, ha come default il caricamento del contenuto associato all'URL indicata nell'attributo *href*.
- Quando un evento ha un'azione di default, il valore della sua proprietà cancelable è settato a true.

Eventi cancellabili – 2

- Si può evitare l'esecuzione del comportamento di default invocando il metodo preventDefault()
- Il flusso dell'evento non viene interrotto, le tre fasi saranno comunque eseguite
- Alla fine della fase di *bubbling* l'azione di default non sarà eseguita

Esempio

```
<div id="d1">
  <a href="http://apress.com">Sito Apress</a>
  <a href="http://w3c.org">Sito W3C</a>
</div>
```

```
var elems = document.querySelectorAll("a");
for (var i = 0; i < elems.length; i++) {
      elems[i].addEventListener("click", handleClick, false);
}

var elem = document.getElementById('d1');
elem.addEventListener("click", saluta, true);
elem.addEventListener("click", saluta, false);
```

```
function handleClick(e) {
 if (!confirm("Vuoi visitare il sito " + e.target.href + " ?"))
                    e.preventDefault();
}

function saluta(e){
if (e.eventPhase != Event.AT_TARGET)
   alert("Esecuzione funzione saluta\n" +
          "Sono nella fase '" +
          getPhaseName(e.eventPhase) +
          "'\n Sono nell'elemento " +
          e.currentTarget.nodeName);
}
```

Gestori di eventi e moduli

- onchange
  - Attivato quando l'utente lascia una casella di testo
- onkeydown
  - Attivato quando l'utente preme un tasto
- onkeyup
  - Attivato quando l'utente un rilascia il tasto
- onkeypress
  - Attivato tra onkeydown e onkeyup
- ondblclick
  - Attivato quando l'utente preme due volte il tasto sinistro del mouse

Eventi del mouse

- click
- dblclick
- mousedown
- mouseenter
- mouseleave
- mousemove
- mouseout
- mouseover
- mouseup

Quando uno di questi eventi è scatenato il browser inizializza un oggetto MouseEvent che estende l'oggetto Event

Oggetto MouseEvent

- button
  - Vale 0 (tasto sinistro) – 1 (tasto centrale)
  - 2 (tasto destro)
- altKey / shiftKey / ctrlKey
  - Vale true se alt / shift / ctrl è stato premuto
- clientX / clientY
  - Posizione x / y del cursore rispetto al viewport
- screenX / screenY
  - Posizione x / y del cursore rispetto allo schermo

Eventi associati al focus

- blur
  - L'elemento ha perso il focus
- focus
  - L'elemento ha ottenuto il focus
- focusin
  - L'elemento sta per ricevere il focus
- focusout
  - L'elemento sta per perdere il focus

Quando uno di questi eventi è scatenato il browser inizializza un oggetto FocusEvent che estende l'oggetto Event

Eventi della tastiera

- keydown
- keypress
- keyup

Quando uno di questi eventi è scatenato il browser inizializza un oggetto KeyboardEvent che estende l'oggetto FocusEvent

Oggetto KeyboardEvent – 1

- keyCode
  - Codice numerico che rappresenta il tasto premuto dipende dall'implementazione e dal sistema
  - Deprecato usare key se supportato
- charCode
  - Codice UNICODE del tasto premuto
  - Possiamo ottenere il carattere con String.fromCharCode(e.charCode)

Oggetto KeyboardEvent – 2

- ctrlKey / shiftKey / altKey
  - Vale true se il tasto ctrl / shift / alt è stato premuto
- repeat
  - Vale true se il tasto continua ad essere premuto