JavaScript Events Handling

13.1 Introduction

- JavaScript events
  - Allow scripts to respond to user interactions and modify the page accordingly
- Events and event handling
  - Help make web applications more dynamic and interactive

13.2 Reviewing the load Event

- The window object's load event fires when the window finishes loading successfully
  - All its children are loaded and all external files referenced by the page are loaded
- Every DOM element has a load event, but it's most commonly used on the window object
- The next example reviews the load event
- The load event's handler creates an interval timer that updates a span with the number of seconds that have elapsed since the document was loaded
  - The document's paragraph contains the span
  - http://www.w3schools.com/jsref/met_win_setinterval.asp

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 13.1: onload.html -->
4  <!-- Demonstrating the load event. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>load Event</title>
9        <link rel = "stylesheet" type = "text/css" href = "style.css">
10       <script src = "load.js"></script>
11    </head>
12    <body>
13       <p>Seconds you have spent viewing this page so far:
14       <span id = "soFar">0</span></p>
15    </body>
16 </html>
```

Fig. 13.1 | Demonstrating the `window`'s `load` event. (Part 1 of 2.)

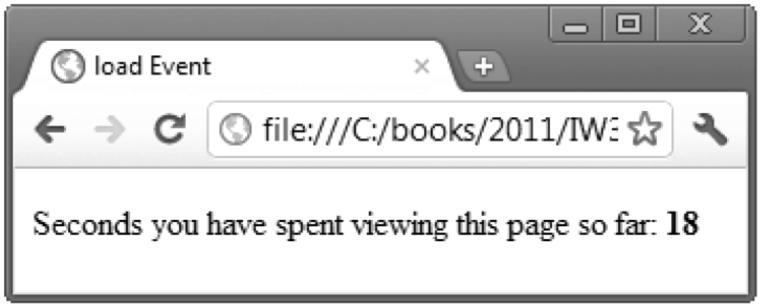

Fig. 13.1 | Demonstrating the `window`'s `load` event. (Part 2 of 2.)

```
1  // Fig. 13.2: load.js
2  // Script to demonstrate the load event.
3  var seconds = 0;
4
5  // called when the page loads to begin the timer
6  function startTimer()
7  {
8     window.setInterval( "updateTime()", 1000 );
9  } // end function startTimer
10
11 // called every 1000 ms to update the timer
12 function updateTime()
13 {
14    ++seconds;
15    document.getElementById( "soFar" ).innerHTML = seconds;
16 } // end function updateTime
17
18 window.addEventListener( "load", startTimer, false );
```

Fig. 13.2 | Script that registers `window`'s `load` event handler and handles the event.

13.2 Reviewing the load Event (Cont.)

- An event handler is a function that responds to an event
- Assigning an event handler to an event on a DOM node is called registering an event handler
- Method **addEventListener** can be called multiple times on a DOM node to register more than one event-handling method for an event
- It's also possible to remove an event listener by calling **removeEventListener** with the same arguments that you passed to **addEventListener** to register the event handler
- If a script in the **head** attempts to get a DOM node for an HTML element in the body **getElementById** returns null
  - Because the body has **not** yet loaded

13.2 Reviewing the load Event (Cont.)

- Two models for registering event handlers
  - Inline model treats events as attributes of HTML elements
  - Traditional model assigns the name of the function to the event property of a DOM node
- The inline model places calls to JavaScript functions directly in HTML code
- The following code indicates that JavaScript function start should be called when the body element loads
  - **<body onload = "start()">**

13.2 Reviewing the load Event (Cont.)

- The traditional model uses a property of an object to specify an event handler
- The following JavaScript code indicates that function start should be called when document loads
  - **document.onload = "start()";**

13.3 Event mouseMove and the event Object

- **mousemove** event occurs whenever the user moves the mouse over the web page
- The next example creates a simple drawing program that allows the user to draw inside a table element in red or blue by holding down the Shift key or Ctrl key and moving the mouse over the box
  - **ctrlKey** property contains a boolean which reflects whether the Ctrl key was pressed during the event
  - **shiftKey** property reflects whether the Shift key was pressed during the event

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 13.3: draw.html -->
4  <!-- A simple drawing program. -->
5  <html>
6     <head>
7        <meta charset="utf-8">
8        <title>Simple Drawing Program</title>
9        <link rel = "stylesheet" type = "text/css" href = "style.css">
10       <script src = "draw.js"></script>
11    </head>
12    <body>
13       <table id = "canvas">
14          <caption>Hold <em>Ctrl</em> (or <em>Control</em>) to draw blue.
15             Hold <em>Shift</em> to draw red.</caption>
16          <tbody id = "tablebody"></tbody>
17       </table>
18    </body>
19 </html>
```

Fig. 13.3 | Simple drawing program. (Part 1 of 3.)

a) User holds the *Shift* key and moves the mouse to draw in red.

Fig. 13.3 | Simple drawing program. (Part 2 of 3.)

b) User holds the *Ctrl* key and moves the mouse to draw in blue.

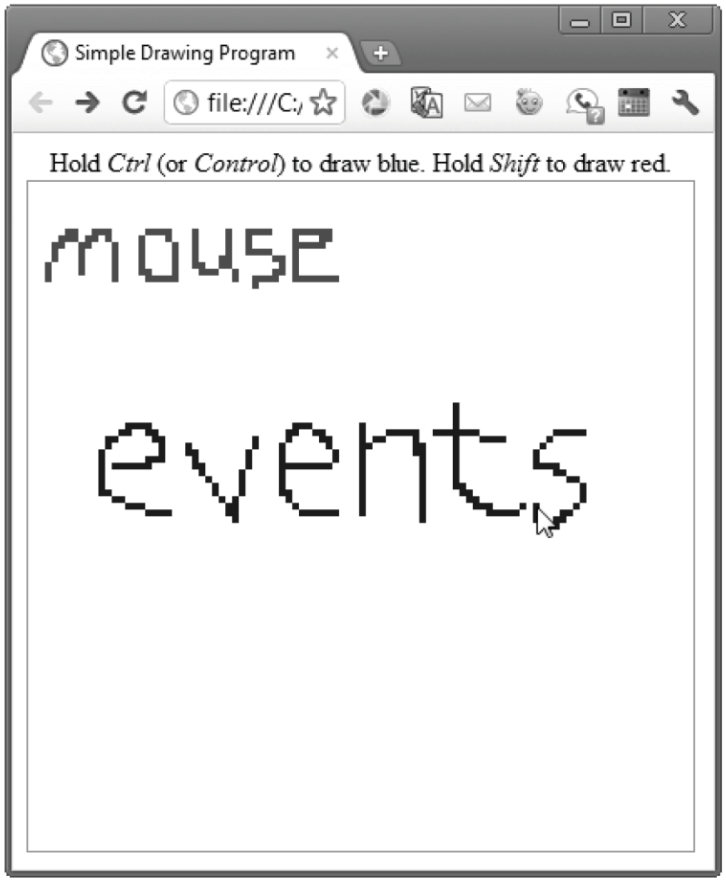

Fig. 13.3 | Simple drawing program. (Part 3 of 3.)

```
1  // Fig. 13.4: draw.js
2  // A simple drawing program.
3  // initialization function to insert cells into the table
4  function createCanvas()
5  {
6     var side = 100;
7     var tbody = document.getElementById( "tablebody" );
8
9     for ( var i = 0; i < side; ++i )
10    {
11       var row = document.createElement( "tr" );
12
13       for ( var j = 0; j < side; ++j )
14       {
15          var cell = document.createElement( "td" );
16          row.appendChild( cell );
17       } // end for
18
19       tbody.appendChild( row );
20    } // end for
21
```

Fig. 13.4 | JavaScript code for the simple drawing program. (Part 1 of 2.)

```
22       // register mousemove listener for the table
23       document.getElementById( "canvas" ).addEventListener(
24          "mousemove", processMouseMove, false );
25    } // end function createCanvas
26
27    // processes the onmousemove event
28    function processMouseMove( e )
29    {
30       if ( e.target.tagName.toLowerCase() == "td" )
31       {
32          // turn the cell blue if the Ctrl key is pressed
33          if ( e.ctrlKey )
34          {
35             e.target.setAttribute( "class", "blue" );
36          } // end if
37
38          // turn the cell red if the Shift key is pressed
39          if ( e.shiftKey )
40          {
41             e.target.setAttribute( "class", "red" );
42          } // end if
43       } // end if
44    } // end function processMouseMove
45
46    window.addEventListener( "load", createCanvas, false );
```

Fig. 13.4 | JavaScript code for the simple drawing program. (Part 2 of 2.)

Property	Description
`altKey`	This value is `true` if the *Alt* key was pressed when the event fired.
`cancelBubble`	Set to `true` to prevent the event from bubbling. Defaults to `false`. (See Section 13.7, Event Bubbling.)
`clientX` and `clientY`	The coordinates of the mouse cursor inside the client area (i.e., the active area where the web page is displayed, excluding scrollbars, navigation buttons, etc.).
`ctrlKey`	This value is `true` if the *Ctrl* key was pressed when the event fired.
`keyCode`	The ASCII code of the key pressed in a keyboard event. See Appendix D for more information on the ASCII character set.
`screenX` and `screenY`	The coordinates of the mouse cursor on the screen coordinate system.
`shiftKey`	This value is `true` if the *Shift* key was pressed when the event fired.
`target`	The DOM object that received the event.
`type`	The name of the event that fired.

Fig. 13.5 | Some `event`-object properties.

13.3 Event mouseMove and the event Object

- http://www.w3schools.com/jsref/met_document_createelement.asp
- http://www.w3schools.com/jsref/met_node_appendchild.asp
- http://www.w3schools.com/jsref/met_element_setattribute.asp

13.4 Rollovers with mouseover and mouseout

- When the mouse cursor enters an element, an **mouseover** event occurs for that element
- When the mouse cursor leaves the element, a **mouseout** event occurs for that element
- Creating an Image object and setting its **src** property preloads the image

Performance Tip 13.1

Preloading images used in rollover effects prevents a delay the first time an image is displayed.

```
<!DOCTYPE html>

<!-- Fig 13.6: mouseoverout.html -->
<!-- Events mouseover and mouseout. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Events mouseover and mouseout</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "mouseoverout.js"></script>
   </head>
   <body>
      <h1><img src = "heading1.png" id = "heading"
         alt = "Heading Image"></h1>
      <p>Can you tell a color from its hexadecimal RGB code
      value? Look at the hex code, guess its color. To see
      what color it corresponds to, move the mouse over the
      hex code. Moving the mouse out of the hex code's table
      cell will display the color name.</p>
```

Fig. 13.6 | HTML5 document to demonstrate mouseover and mouseout. (Part 1 of 6.)

```
20          <div>
21             <ul>
22                <li id = "Black">#000000</li>
23                <li id = "Blue">#0000FF</li>
24                <li id = "Magenta">#FF00FF</li>
25                <li id = "Gray">#808080</li>
26                <li id = "Green">#008000</li>
27                <li id = "Lime">#00FF00</li>
28                <li id = "Maroon">#800000</li>
29                <li id = "Navy">#000080</li>
30                <li id = "Olive">#808000</li>
31                <li id = "Purple">#800080</li>
32                <li id = "Red">#FF0000</li>
33                <li id = "Silver">#C0C0C0</li>
34                <li id = "Cyan">#00FFFF</li>
35                <li id = "Teal">#008080</li>
36                <li id = "Yellow">#FFFF00</li>
37                <li id = "White">#FFFFFF</li>
38             </ul>
39          </div>
40       </body>
41    </html>
```

Fig. 13.6 | HTML5 document to demonstrate `mouseover` and `mouseout`. (Part 2 of 6.)

a) The page loads with the blue heading image and all the hex codes in black.

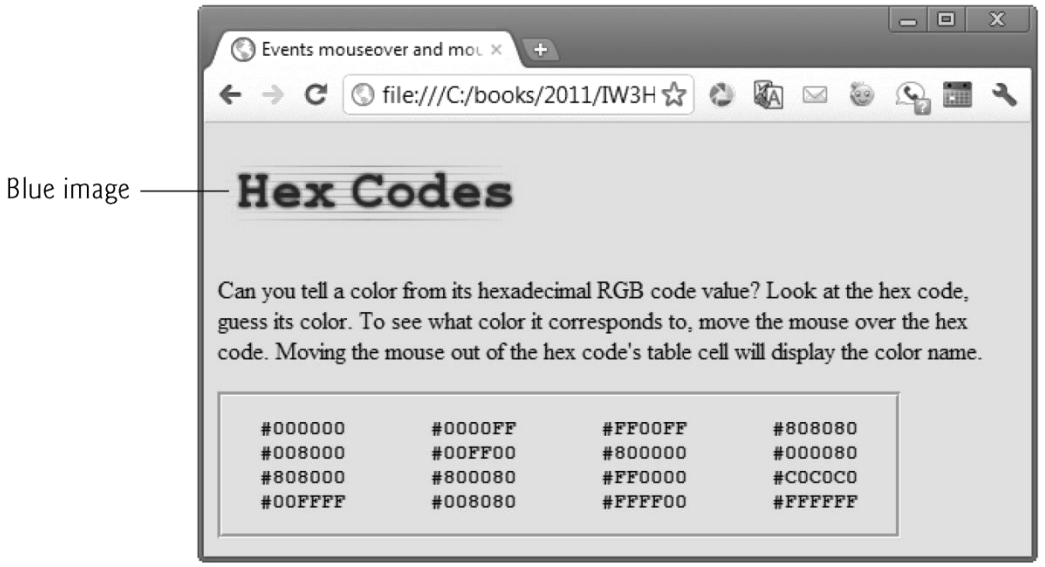

Fig. 13.6 | HTML5 document to demonstrate `mouseover` and `mouseout`. (Part 3 of 6.)

b) The heading image switches to an image with green text when the mouse rolls over it.

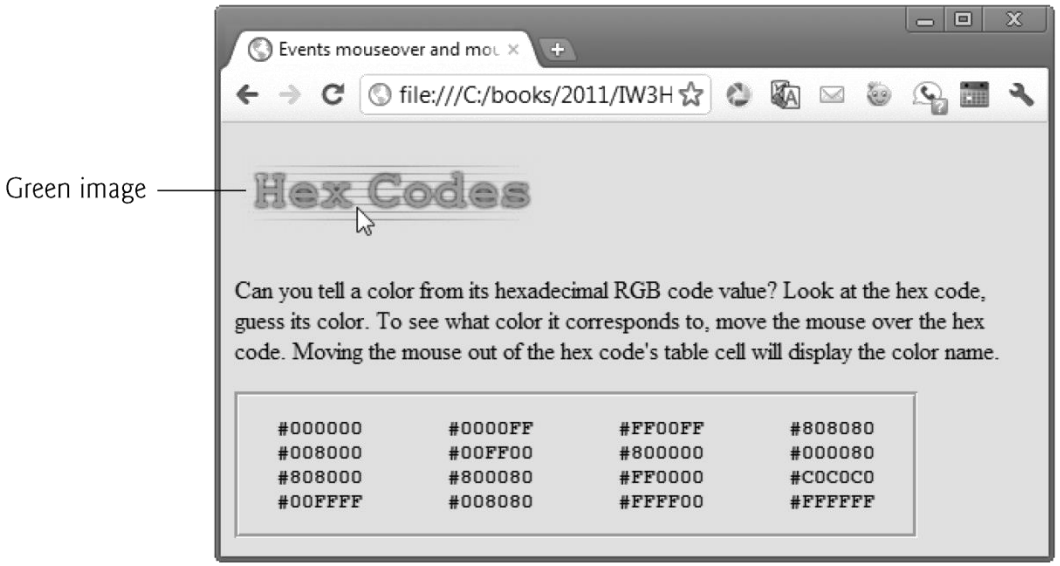

Fig. 13.6 | HTML5 document to demonstrate `mouseover` and `mouseout`. (Part 4 of 6.)

c) When mouse rolls over a hex code, the text color changes to the color represented by the hex code. Notice that the heading image has become blue again because the mouse is no longer over it.

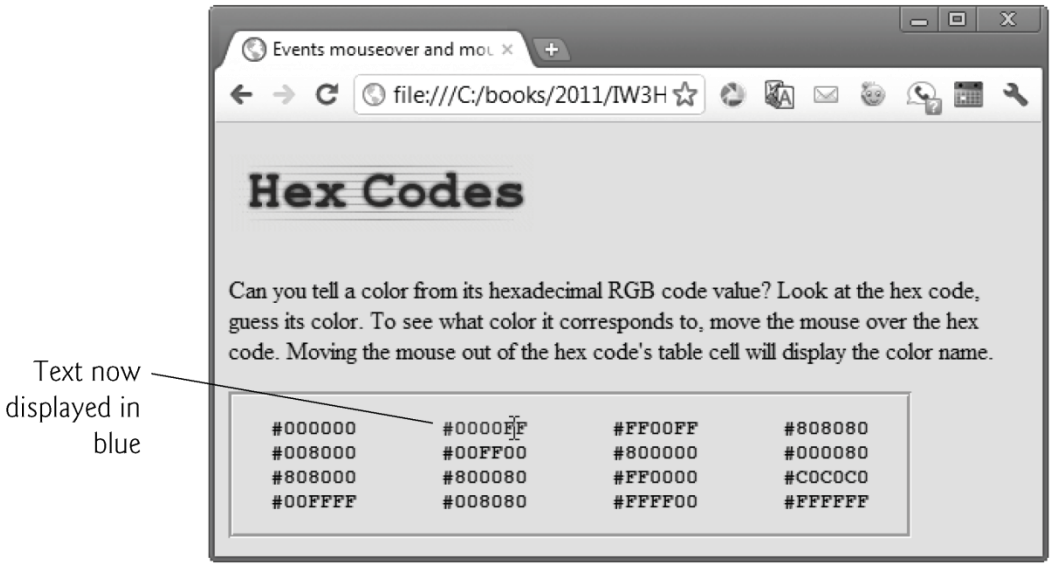

Fig. 13.6 | HTML5 document to demonstrate `mouseover` and `mouseout`. (Part 5 of 6.)

d) When the mouse leaves the hex code's table cell, the text changes to the name of the color.

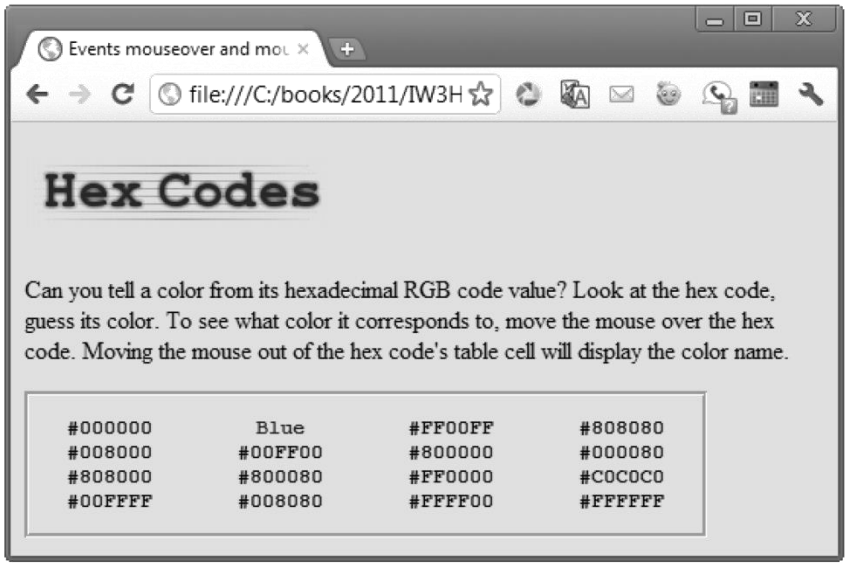

Fig. 13.6 | HTML5 document to demonstrate `mouseover` and `mouseout`. (Part 6 of 6.)

```
1  // Fig 13.7: mouseoverout.js
2  // Events mouseover and mouseout.
3  image1 = new Image();
4  image1.src = "heading1.png";
5  image2 = new Image();
6  image2.src = "heading2.png";
7
8  function mouseOver( e )
9  {
10    // swap the image when the mouse moves over it
11    if ( e.target.getAttribute( "id" ) == "heading" )
12    {
13       e.target.setAttribute( "src", image2.getAttribute( "src" ) );
14    } // end if
15
16    // if the element is an li, assign its id to its color
17    // to change the hex code's text to the corresponding color
18    if ( e.target.tagName.toLowerCase() == "li"  )
19    {
20       e.target.setAttribute( "style",
21          "color: " + e.target.getAttribute( "id" ) );
22    } // end if
23 } // end function mouseOver
```

Fig. 13.7 | Processing the `mouseover` and `mouseout` events. (Part 1 of 2.)

```
24
25  function mouseOut( e )
26  {
27     // put the original image back when the mouse moves away
28     if ( e.target.getAttribute( "id" ) == "heading" )
29     {
30        e.target.setAttribute( "src", image1.getAttribute( "src" ) );
31     } // end if
32
33     // if the element is an li, assign its id to innerHTML
34     // to display the color name
35     if ( e.target.tagName.toLowerCase() == "li"  )
36     {
37        e.target.innerHTML = e.target.getAttribute( "id" );
38     } // end if
39  } // end function mouseOut
40
41  document.addEventListener( "mouseover", mouseOver, false );
42  document.addEventListener( "mouseout", mouseOut, false );
```

Fig. 13.7 | Processing the `mouseover` and `mouseout` events. (Part 2 of 2.)

13.5 Form Processing with focus and blur

- **focus** event fires when an element gains focus
  - When the user clicks a form field or uses the Tab key to move between form elements
- **blur** fires when an element loses focus
  - When another control gains the focus

```
<!DOCTYPE html>

<!-- Fig. 13.8: focusblur.html -->
<!-- Demonstrating the focus and blur events. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>A Form Using focus and blur</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "focusblur.js"></script>
   </head>
   <body>
      <form id = "myForm" action = "">
         <p><label class = "fixed" for = "name">Name:</label>
            <input type = "text" id =  "name"
               placeholder = "Enter name"></p>
         <p><label class = "fixed" for = "email">E-mail:</label>
            <input type = "email" id = "email"
               placeholder = "Enter e-mail address"></p>
         <p><label>Click here if you like this site
            <input type = "checkbox" id =  "like"></label></p>
         <p><label for = "comments">Any comments?</p>
         <textarea id =  "comments"
            placeholder = "Enter comments here"></textarea>
```

Fig. 13.8 | Demonstrating the `focus` and `blur` events. (Part 1 of 3.)

```
25             <p><input id = "submit" type = "submit">
26                <input id = "reset" type = "reset"></p>
27          </form>
28          <p id = "helpText"></p>
29       </body>
30    </html>
```

a) The blue message at the bottom of the page instructs the user to enter a name when the **Name:** field has the focus.

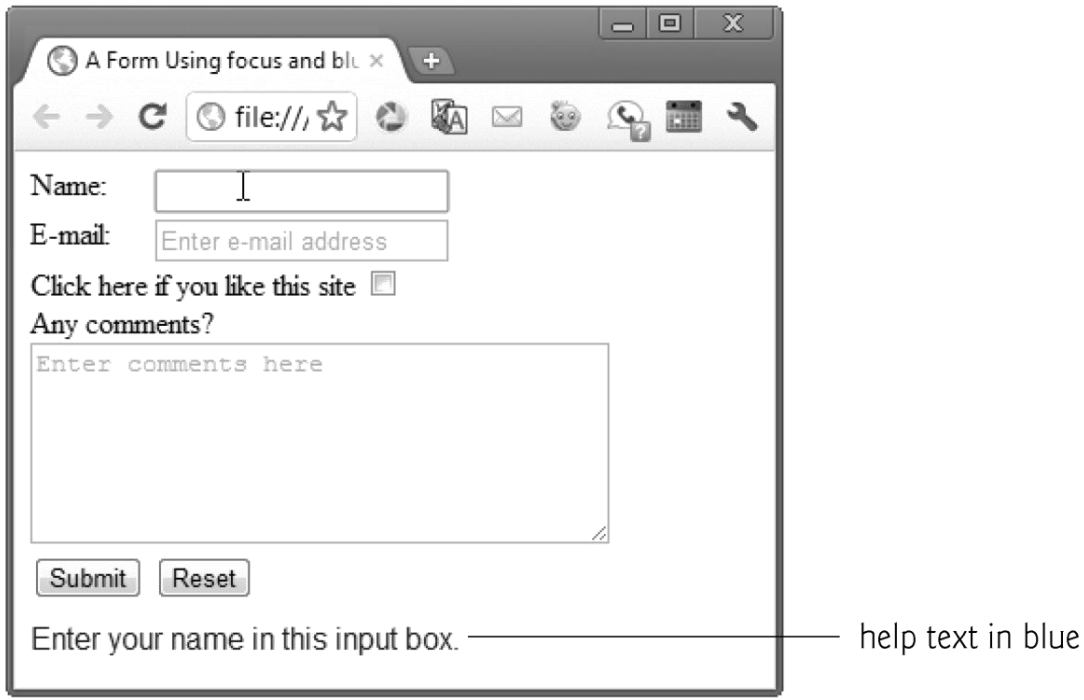

Fig. 13.8 | Demonstrating the `focus` and `blur` events. (Part 2 of 3.)

b) The message changes depending on which field has focus—this window shows the help text for the `comments` `textarea`.

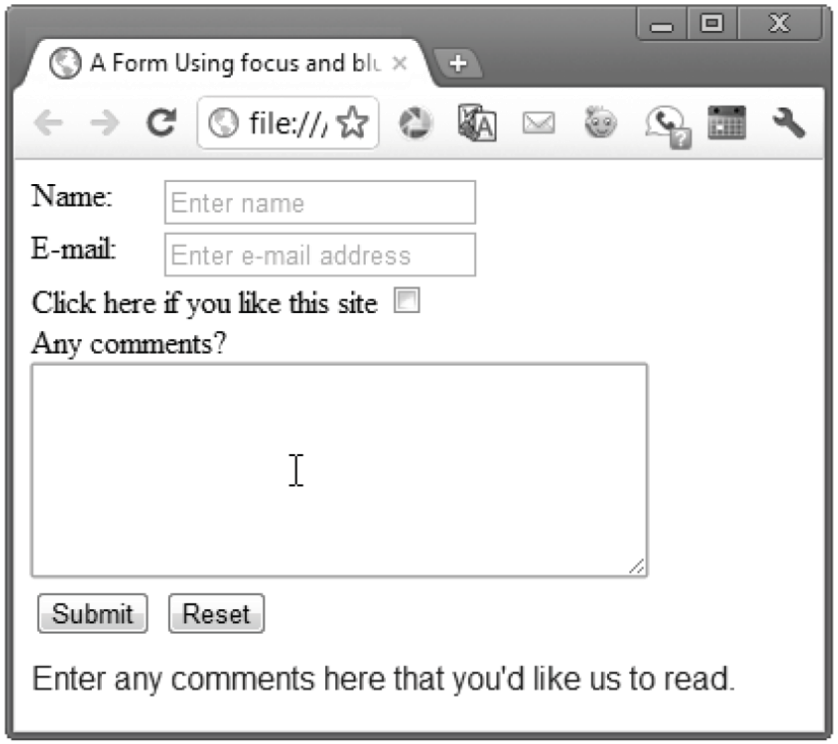

Fig. 13.8 | Demonstrating the `focus` and `blur` events. (Part 3 of 3.)

```
1  // Fig. 13.9: focusblur.js
2  // Demonstrating the focus and blur events.
3  var helpArray = [ "Enter your name in this input box.",
4    "Enter your e-mail address in the format user@domain.",
5    "Check this box if you liked our site.",
6    "Enter any comments here that you'd like us to read.",
7    "This button submits the form to the server-side script.",
8    "This button clears the form.", "" ];
9  var helpText;
10
11 // initialize helpTextDiv and register event handlers
12 function init()
13 {
14    helpText = document.getElementById( "helpText" );
15
16    // register listeners
17    registerListeners( document.getElementById( "name" ), 0 );
18    registerListeners( document.getElementById( "email" ), 1 );
19    registerListeners( document.getElementById( "like" ), 2 );
20    registerListeners( document.getElementById( "comments" ), 3 );
21    registerListeners( document.getElementById( "submit" ), 4 );
22    registerListeners( document.getElementById( "reset" ), 5 );
23 } // end function init
24
```

Fig. 13.9 | Demonstrating the `focus` and `blur` events. (Part 1 of 2.)

```
25  // utility function to help register events
26  function registerListeners( object, messageNumber )
27  {
28     object.addEventListener( "focus",
29        function() { helpText.innerHTML = helpArray[ messageNumber ]; },
30        false );
31     object.addEventListener( "blur",
32        function() { helpText.innerHTML = helpArray[ 6 ]; }, false );
33  } // end function registerListener
34
35  window.addEventListener( "load", init, false );
```

Fig. 13.9 | Demonstrating the `focus` and `blur` events. (Part 2 of 2.)

13.6 More Form Processing with submit and reset

- **submit** and **reset** events fire when a form is submitted or reset, respectively
- The anonymous function executes in response to the user's submitting the form by clicking the Submit button or pressing the Enter key
- **confirm** method asks the users a question, presenting them with an OK button and a Cancel button
  - If the user clicks OK, confirm returns true; otherwise, confirm returns false
- By returning either true or false, event handlers dictate whether the default action for the event is taken

```
1  // Fig. 13.10: focusblur.js
2  // Demonstrating the focus and blur events.
3  var helpArray = [ "Enter your name in this input box.",
4    "Enter your e-mail address in the format user@domain.",
5    "Check this box if you liked our site.",
6    "Enter any comments here that you'd like us to read.",
7    "This button submits the form to the server-side script.",
8    "This button clears the form.", "" ];
9  var helpText;
10
11 // initialize helpTextDiv and register event handlers
12 function init()
13 {
14    helpText = document.getElementById( "helpText" );
15
16    // register listeners
17    registerListeners( document.getElementById( "name" ), 0 );
18    registerListeners( document.getElementById( "email" ), 1 );
19    registerListeners( document.getElementById( "like" ), 2 );
20    registerListeners( document.getElementById( "comments" ), 3 );
21    registerListeners( document.getElementById( "submit" ), 4 );
22    registerListeners( document.getElementById( "reset" ), 5 );
23
24    var myForm = document.getElementById( "myForm" );
```

Fig. 13.10 | Demonstrating the `focus` and `blur` events. (Part 1 of 3.)

```
25       myForm.addEventListener( "submit",
26          function()
27          {
28             return confirm( "Are you sure you want to submit?" );
29          }, // end anonymous function
30          false );
31       myForm.addEventListener( "reset",
32          function()
33          {
34             return confirm( "Are you sure you want to reset?" );
35          }, // end anonymous function
36          false );
37    } // end function init
38
39    // utility function to help register events
40    function registerListeners( object, messageNumber )
41    {
42       object.addEventListener( "focus",
43          function() { helpText.innerHTML = helpArray[ messageNumber ]; },
44          false );
45       object.addEventListener( "blur",
46          function() { helpText.innerHTML = helpArray[ 6 ]; }, false );
47    } // end function registerListener
48
49    window.addEventListener( "load", init, false );
```

Fig. 13.10 | Demonstrating the `focus` and `blur` events. (Part 2 of 3.)

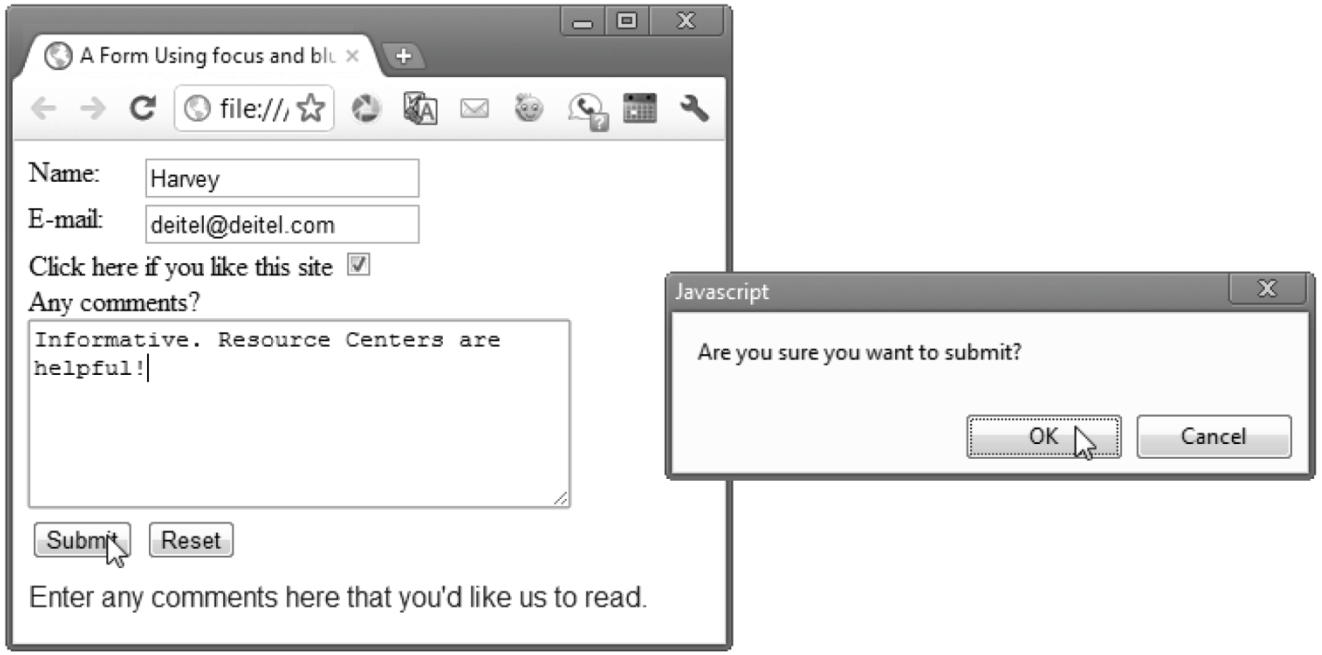

Fig. 13.10 | Demonstrating the `focus` and `blur` events. (Part 3 of 3.)

13.7 Event Bubbling

- Event bubbling
  - The process whereby events fired on child elements "bubble" up to their parent elements
  - When an event is fired on an element, it is first delivered to the element's event handler (if any)
    - Then to the parent element's event handler (if any)
- If you intend to handle an event in a child element alone, you should cancel the bubbling of the event in the child element's event-handling code by using the **cancelBubble** property of the event object

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 13.11: bubbling.html -->
4  <!-- Canceling event bubbling. -->
5  <html>
6     <head>
7        <meta charset="utf-8">
8        <title>Event Bubbling</title>
9        <script src = "bubbling.js">
10    </head>
11    <body>
12       <p id = "bubble">Bubbling enabled.</p>
13       <p id = "noBubble">Bubbling disabled.</p>
14    </body>
15 </html>
```

Fig. 13.11 | Canceling event bubbling. (Part 1 of 3.)

a) User clicks the first paragraph, for which bubbling is enabled.

b) Paragraph's event handler causes an alert.

c) Document's event handler causes another alert, because the event bubbles up to the document.

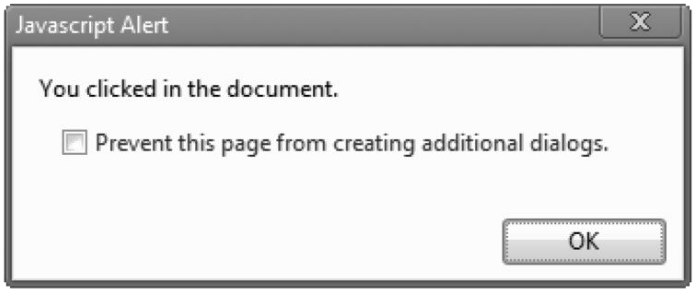

Fig. 13.11 | Canceling event bubbling. (Part 2 of 3.)

d) User clicks the second paragraph, for which bubbling is disabled.

e) Paragraph's event handler causes an alert. The document's event handler is not called.

Fig. 13.11 | Canceling event bubbling. (Part 3 of 3.)

Common Programming Error 13.1

Forgetting to cancel event bubbling when necessary may cause unexpected results in your scripts.

```
1  // Fig. 13.12: bubbling.js
2  // Canceling event bubbling.
3  function documentClick()
4  {
5     alert( "You clicked in the document." );
6  } // end function documentClick
7
8  function bubble( e )
9  {
10    alert( "This will bubble." );
11    e.cancelBubble = false;
12 } // end function bubble
13
14 function noBubble( e )
15 {
16    alert( "This will not bubble." );
17    e.cancelBubble = true;
18 } // end function noBubble
19
```

Fig. 13.12 | Canceling event bubbling. (Part 1 of 2.)

```
20   function registerEvents()
21   {
22      document.addEventListener( "click", documentClick, false );
23      document.getElementById( "bubble" ).addEventListener(
24         "click", bubble, false );
25      document.getElementById( "noBubble" ).addEventListener(
26         "click", noBubble, false );
27   } // end function registerEvents
28
29   window.addEventListener( "load", registerEvents, false );
```

Fig. 13.12 | Canceling event bubbling. (Part 2 of 2.)

13.8 More Events

- The following slide lists some common events and their descriptions
- The actual DOM event names begin with "on", but we show the names you use with **addEventListener** here
  - http://www.w3schools.com/jsref/dom_obj_event.asp

Event	Description
`abort`	Fires when image transfer has been interrupted by user.
`change`	Fires when a new choice is made in a `select` element, or when a text input is changed and the element loses focus.
`click`	Fires when the user clicks the mouse.
`dblclick`	Fires when the user double clicks the mouse.
`focus`	Fires when a form element gets the focus.
`keydown`	Fires when the user pushes down a key.
`keypress`	Fires when the user presses then releases a key.
`keyup`	Fires when the user releases a key.
`load`	Fires when an element and all its children have loaded.
`mousedown`	Fires when a mouse button is pressed.
`mousemove`	Fires when the mouse moves.
`mouseout`	Fires when the mouse leaves an element.
`mouseover`	Fires when the mouse enters an element.

Fig. 13.13 | Common events. (Part 1 of 2.)

Event	Description
`mouseup`	Fires when a mouse button is released.
`reset`	Fires when a form resets (i.e., the user clicks a reset button).
`resize`	Fires when the size of an object changes (i.e., the user resizes a window or frame).
`select`	Fires when a text selection begins (applies to `input` or `textarea`).
`submit`	Fires when a form is submitted.
`unload`	Fires when a page is about to unload.

Fig. 13.13 | Common events. (Part 2 of 2.)