Topic 5 JavaScript Event Handling
CITS3403 Agile Web Development

Reference: Sebesta, Chapter 5 and 6
Event-Driven Programming

• *Event-driven programming* or *event-based programming*
  
  – programming paradigm in which the flow of the program is determined by *sensor outputs* or *user actions* (*mouse* clicks, key presses) or *messages from other programs*
  
  – not new - from hardware interrupts to multi-process operating systems to distributed programming to Java listeners to Exceptions...

• *Fundamental to web-based programming*
  
  – client-server model
  
  – stateless programming
  
  – controlled from browser (user) end
Event-Driven Programming

• **Batch program**

  read a number (from the keyboard) and store it in variable A[0]
  read a number (from the keyboard) and store it in variable A[1]

  – *synchronous* (program waits for input)

• **Event-driven program**

  set counter K to 0
  repeat {  
    if a number has been entered (from the keyboard) {    
      store in A[K] and increment K    
      if K equals 2 print A[0]+A[1] and reset K to 0    
    }
  }

Event-Driven Programming

• Program “loop” divided into two distinct tasks
  – event detection
  – event handling

• Application programmer may be freed from event detection (and hence loop) in a number of ways
  – embedded programs may use interrupts - handled by hardware (no loop needed)
  – programming environment or execution environment may do this for you - in our case the browser
    ➡ allows programmer to focus on event handling
Event Handling

- Browser “listens” (polls or interrupts) for events
  - user actions (eg. <enter>, mouse clicks, ...)
  - server responses (eg. page loaded, AJAX responses, calculation, ...)
- When it recognises an event, it invokes the appropriate code to handle the event (*event handler*), passing information about the event as required
- But how does the browser know what code to call?
Event Registration (Binding)

- For the browser to know what code to invoke for different actions, code elements must be *registered* with, or *bound* to, events.

- What defines the events, their meanings, and parameters?
  - the DOM!
History

• HTML 4.0 Standard → first specification of event model for documents
  – referred to as DOM 0 event model
  – supported by all browsers supporting JavaScript
  – limited scope

• DOM 2 event model
  – comprehensive
  – supported by FX2 and above
  – not supported by IE7
Event Registration

• DOM 0 provides two ways to register an event handler:

1. Assign the event handler script to an *event tag attribute*

   <input type = "button" id = "myButton"
       onclick = "alert('You clicked my button!');" />

   *onclick* is a tag attribute for the button “click” event

Usually the handler script is more than a single statement and called as a function:

   <input type = "button" id = "myButton"
       onclick = "myButtonHandler();" />
# Events and their Tag Attributes

<table>
<thead>
<tr>
<th>Event</th>
<th>Tag Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>blur</td>
<td>onblur</td>
</tr>
<tr>
<td>change</td>
<td>onchange</td>
</tr>
<tr>
<td>click</td>
<td>onclick</td>
</tr>
<tr>
<td>dblclick</td>
<td>ondblclick</td>
</tr>
<tr>
<td>focus</td>
<td>onfocus</td>
</tr>
<tr>
<td>keydown</td>
<td>onkeydown</td>
</tr>
<tr>
<td>keypress</td>
<td>onkeypress</td>
</tr>
<tr>
<td>keyup</td>
<td>onkeyup</td>
</tr>
<tr>
<td>load</td>
<td>onload</td>
</tr>
<tr>
<td>mousedown</td>
<td>onmousedown</td>
</tr>
<tr>
<td>mousemove</td>
<td>onmousemove</td>
</tr>
<tr>
<td>mouseout</td>
<td>onmouseout</td>
</tr>
<tr>
<td>mouseover</td>
<td>onmouseover</td>
</tr>
<tr>
<td>mouseup</td>
<td>onmouseup</td>
</tr>
<tr>
<td>reset</td>
<td>onreset</td>
</tr>
<tr>
<td>select</td>
<td>onselect</td>
</tr>
<tr>
<td>submit</td>
<td>onsubmit</td>
</tr>
<tr>
<td>unload</td>
<td>onunload</td>
</tr>
</tbody>
</table>

Tag Attributes and their Tags

• Most event tag attributes can appear in several tags

• Meaning (action) depends on both the tag attribute and the tag in which it appears. Eg.
  – an element gains “focus” when the mouse is passed over it and left clicked, or user tabs to element
  – lose focus when it passes to another element - called blurring
  ▸ different meaning (action) for <a> and <textarea>
## Tag Attributes and their Tags

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onblur</td>
<td><code>&lt;a&gt;</code></td>
<td>The link loses the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;button&gt;</code></td>
<td>The button loses the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;input&gt;</code></td>
<td>The input element loses the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;textarea&gt;</code></td>
<td>The text area loses the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;select&gt;</code></td>
<td>The selection element loses the input focus</td>
</tr>
<tr>
<td>onchange</td>
<td><code>&lt;input&gt;</code></td>
<td>The input element is changed and loses the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;textarea&gt;</code></td>
<td>The text area is changed and loses the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;select&gt;</code></td>
<td>The selection element is changed and loses the input focus</td>
</tr>
<tr>
<td>onclick</td>
<td><code>&lt;a&gt;</code></td>
<td>The user clicks on the link</td>
</tr>
<tr>
<td></td>
<td><code>&lt;input&gt;</code></td>
<td>The input element is clicked</td>
</tr>
<tr>
<td>ondblclick</td>
<td>Most elements</td>
<td>The user double clicks the left mouse button</td>
</tr>
<tr>
<td>onfocus</td>
<td><code>&lt;a&gt;</code></td>
<td>The link acquires the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;input&gt;</code></td>
<td>The input element receives the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;textarea&gt;</code></td>
<td>A text area receives the input focus</td>
</tr>
<tr>
<td></td>
<td><code>&lt;select&gt;</code></td>
<td>A selection element receives the input focus</td>
</tr>
<tr>
<td>onkeydown</td>
<td><code>&lt;body&gt;</code>, form elements</td>
<td>A key is pressed down</td>
</tr>
<tr>
<td>onkeypress</td>
<td><code>&lt;body&gt;</code>, form elements</td>
<td>A key is pressed down and released</td>
</tr>
<tr>
<td>onkeyup</td>
<td><code>&lt;body&gt;</code>, form elements</td>
<td>A key is released</td>
</tr>
<tr>
<td>onload</td>
<td><code>&lt;body&gt;</code></td>
<td>The document is finished loading</td>
</tr>
</tbody>
</table>
# Tag Attributes and their Tags

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onmousedown</td>
<td>Most elements</td>
<td>The user clicks the left mouse button</td>
</tr>
<tr>
<td>onmousemove</td>
<td>Most elements</td>
<td>The user moves the mouse cursor within the element</td>
</tr>
<tr>
<td>onmouseout</td>
<td>Most elements</td>
<td>The mouse cursor is moved away from being over the element</td>
</tr>
<tr>
<td>onmouseover</td>
<td>Most elements</td>
<td>The mouse cursor is moved over the element</td>
</tr>
<tr>
<td>onmouseup</td>
<td>Most elements</td>
<td>The left mouse button is unclicked</td>
</tr>
<tr>
<td>onreset</td>
<td>&lt;form&gt;</td>
<td>The reset button is clicked</td>
</tr>
<tr>
<td>onselect</td>
<td>&lt;input&gt;</td>
<td>The mouse cursor is moved over the element</td>
</tr>
<tr>
<td></td>
<td>&lt;textarea&gt;</td>
<td>The text area is selected within the text area</td>
</tr>
<tr>
<td>onsubmit</td>
<td>&lt;form&gt;</td>
<td>The Submit button is pressed</td>
</tr>
<tr>
<td>onunload</td>
<td>&lt;body&gt;</td>
<td>The user exits the document</td>
</tr>
</tbody>
</table>
Event Registration (cont.)

2. Assign the event handler to the appropriate *property of the element’s object*

```html
<input type = "button" id = "myButton" />

document.getElementById("myButton").onclick = myButtonHandler;
```

- statement must follow both handler function and form element so (JavaScript) interpreter has seen both
- note: just function name, not function call (or string)
Handling Events from Body Elements

```html
<body onload="load_greeting();">
  <p />
</body>

function load_greeting () {
  alert("You are visiting the home page of\n" + "Pete’s Pickled Peppers \n" + "Welcome!!!");
}
```
Handling Events from Button Elements

• An event can be registered for this tag in two ways
  
  \[
  \text{<input type="button" name="freeOffer" id="freeButton"/>}
  \]

• Using an event attribute
  
  \[
  \text{<input type="button" name="freeOffer" id="freeButton" onclick="freebuttonHandler();"/>}
  \]

• Assigning to a property of the element node
  
  \[
  \text{document.getElementById("freeButton").onclick = freeButtonHandler}
  \]
  
  – Note that the function name, a reference to the function, is assigned
  – Writing freeButtonHandler() would assign the return value of the function call as the handler (possible, but unlikely)
Checkboxes and Radio Buttons

• `radio_click.html` displays an alert when a radio button is clicked
  – Note that a parameter is passed to the handler function

• In `radio_click2.html`, a reference to the handler function is assigned to the `onclick` property of each element node
  – Note that no parameters are passed to the function
  – The handler code must identify the element that caused the call
Comparison of Registration Methods

• Assigning to a node property helps separate HTML and code → modularity

• Assigning to a node property allows reassignment later if the handler needs to be changed
Handling Events from Text Box and Password Elements

• By manipulating the focus event the user can be prevented from changing the amount in a text input field
  – Example [nochange.html](#) illustrates ‘blurring’ a field whenever it gains focus

• Note: this is possible to work around
  – Copy the page but leave out the validation code
  – Simulate an HTTP request directly with socket-level programming
  ➤ If the validity of data is important, the server needs to check it
Validating Form Input

• Validity checking on the server requires a round-trip for the server to check the data and then to respond with an appropriate error page

• Validating data using JavaScript provides
  – quicker interaction for the user
  – reduced load on server - usually busier than client
  – less network traffic
Validating Form Input

• Handling a data validity error
  – alert message specifying correct range or format
  – put the focus in the field in question

    `document.getElementById(“phone”).focus();`

  – highlight the text for easier editing

    `document.getElementById(“phone”).select();`
Validating Form Input

• If an event handler returns `false`, default actions are not taken by the browser
  – This can be used in a Submit button event handler to check validity and not submit if there are problems
• Example `pswd_chk.html` illustrates validity checking
Validating Input

- The `validator.html` example demonstrates using regular expressions to validate text input

- The name is First, Last, Middle-Initial, each part capitalized
  
  - `/^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.[?$/`

- The phone is ddd-ddd-dddd where d is a digit
  
  - `/^\d{3}-\d{3}-\d{4}$/`

- Each pattern uses the ^ and $ anchors to make sure the entire string matches
DOM 2 Event Model

- DOM 2 is defined in *modules*
- The *Events* module defines several submodules
  - *HTMLEvents* and *MouseEvents* are common
- An event object is passed as a parameter to an event handler
  - Properties of this object provide information about the event
  - Some event types will extend the interface to include information relevant to the subtype. For example, a mouse event will include the location of the mouse at the time of the event
Event Flow

• DOM 2 defines a process for determining which handlers to execute for a particular event

• The event object representing the event is created at a particular node called the *target node*

• The process has three phases...
Event Flow

• In the **capturing phase** each node from the document root to the target node, in order, is examined.
  – If the node is not the target node and there is a handler for that event at the node and the handler is enabled for capture for the node, the handler is executed
• Then all handlers registered for the target node, if any, are executed

• In the **bubbling phase** each node from the parent of the target node to the root node, in order, is examined
  – If there is a handler for that event at the node and the handler is **not** enabled for capture for the node, the handler is executed
  – Some event types are not allowed to bubble: load, unload, blur and focus among the HTML event types
Event Propagation

- As each handler is executed, properties of the event provide context
  - The `currentTarget` property is the node to which the handler is registered
  - The `target` property is the node to which the event was originally directed
  - `currentTarget` is always the object listening for the event; `target` is the actual target that received the event
- One major advantage of this scheme over DOM 0 is that event handling can be centralized in an ancestor node
- For example, a calculator keyboard will have a number of digit buttons
  - In some GUI frameworks, a handler must be added to each button separately
  - In DOM 2, the buttons could be organized under a single node and the handler placed on the node
Event Handler Registration

- Handlers are called *listeners* in DOM 2

- *addEventListener* is used to register a handler, it takes three parameters
  - A string naming the event type
  - The handler
  - A boolean specifying whether the handler is enabled for the capture phase or not
Example of DOM 2 Event Model

- The validator2.html example modifies the validation example to use DOM 2 style event handling