## JavaScript Execution Environment

- The JavaScript `Window` object represents the window in which the browser displays documents

- The `Window` object provides the largest enclosing referencing environment for scripts

  - Its properties are visible to all scripts in the document (they are the globals)

- Other `Window` properties:

  - `document` - a reference to the `Document` object that the window displays

  - `frames` - an array of references to the frames of the document

  - `forms` - an array of references to the forms of the document

    - Each `Form` object has an `elements` array, which has references to the form's elements

      - Form elements are usually referenced by name, but this is a problem for radio buttons

## The Document Object Model

-Under development by w3c since the mid-90s

- DOM 0 is supported by all JavaScript browsers

- DOM 2 is the latest approved standard

  - Nearly completely supported by NS6
  - IE6's support is lacking some important things

- The DOM is an abstract model that defines the interface between HTML documents and application programs

- It is an OO model - document elements are objects

- A language that supports the DOM must have a binding to the DOM constructs

  - In the JavaScript binding, HTML elements are represented as objects and element attributes are represented as properties

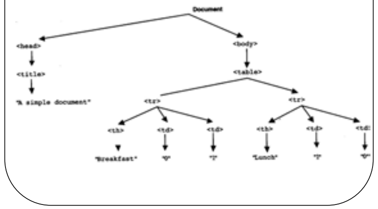
    e.g., `<input type = "text" name = "address">`

    would be represented as an object with two properties, `type` and `name`, with the values `"text"` and `"address"`

→See Figure 5.1

## DOM structure of HTML document

```
<html>
<head> <title> A simple document </title>
</head>
<body>
<table>
    <tr>
        <th> Breakfast </th>
        <td> 0 </td>
        <td> 1 </td>
    </tr>
    <tr>
        <th> Lunch </th>
        <td> 1 </td>
        <td> 0 </td>
    </tr>
 </table>
```



## Element Access in JavaScript

- Example (a document with just one form):

```
<form action = "">
    <input type = "button"  name = "pushMe">
 </form>
```

1. DOM address

   `document.forms[0].element[0]`

   - Problem: A change in the document could invalidate this address

2. Element names – requires the element and all of its ancestors (except `body`) to have `name` attributes

   - Example:

```
<form name = "myForm"  action = "">
    <input type = "button"  name = "pushMe">
 </form>
```

   `document.myForm.pushMe`

   - Problem: Strictly speaking, standard does not allow *form* elements to have names. Still, it is standard practice in many pages.

## Element Access in JavaScript (continued)

3. `getElementById` Method

   - Example:

```
<form action = "">
    <input type = "button"  id = "pushMe">
</form>

document.getElementById("pushMe")
```

To work, the Id must be unique in the document.

## Events and Event Handling

- We look at the DOM 0 event model first

- In event-driven programming, code is executed as a result of a user or browser action

- An *event* is a notification that something specific has occurred, either with the browser or an action of the browser user

- An *event handler* is a script that is implicitly executed in response to the appearance of an event

## Events and Event Handling (continued)

- Because events are JavaScript objects, their names are case sensitive - all are in lowercase only (so click is an event, but Click is not)
- The process of connecting an event handler to an event is called *registration*
- Don't use `document.write` in an event handler, because the output may go on top of the displayed document
- Events (only some are listed here)

| *Event* | *Tag Attribute* |
|---|---|
| `abort` | `onAbort` |
| `blur` | `onBlur` |
| `change` | `onChange` |
| `click` | `onClick` |
| `error` | `onError` |
| `focus` | `onFocus` |
| `load` | `onLoad` |
| `mouseout` | `onMouseOut` |
| `mouseover` | `onMouseOver` |
| `reset` | `onReset` |
| `resize` | `onResize` |
| `select` | `onSelect` |
| `submit` | `onSubmit` |
| `unload` | `onUnload` |

## Events and Event Handling (continued)

- The same attribute can appear in several different tags

e.g., The `onClick` attribute can be in `<a>` and `<input>`

- *A text element gets focus in three ways:*

1. When the user puts the mouse cursor over it and presses the left button

2. When the user tabs to the element

3. By executing the `focus` method

→ See: Table 5.2

# Tags' Events

- The events for each HTML tag are as follows:
- <A>
  - – click (onClick)
  - – mouseOver (onMouseOver)
  - – mouseOut (onMouseOut)
- <AREA>
  - – mouseOver (onMouseOver)
  - – mouseOut (onMouseOut)
- <BODY>
  - – blur (onBlur)
  - – error (onError)
  - – focus (onFocus)
  - – load (onLoad)
  - – unload (onUnload)
- <FORM>
  - – submit (onSubmit)
  - – reset (onReset
- <FRAME>
  - – blur (onBlur)
  - – focus (onFocus)
- <FRAMESET>
  - – blur (onBlur)
  - – error (onError)
  - – focus (onFocus)
  - – load (onLoad)
  - – unload (onUnload)
- <IMG>
  - – abort (onAbort)
  - – error (onError)
  - – load (onLoad)
- <INPUT TYPE = "button">
  - – click (onClick)
- <INPUT TYPE = "checkbox">
  - – click (onClick)
- <INPUT TYPE = "reset">
  - – click (onClick)
- <INPUT TYPE = "submit">
  - – click (onClick)
- <INPUT TYPE = "text">
  - – blur (onBlur)
  - – focus (onFocus)
  - – change (onChange)
  - – select (onSelect)
- <SELECT>
  - – blur (onBlur)
  - – focus (onFocus)
  - – change (onChange)
- <TEXTAREA>
  - – blur (onBlur)
  - – focus (onFocus)
  - – change (onChange)
  - – select (onSelect)
- SEE web page

# Specifying event handlers

- 1. By assigning the event handler script to an event tag attribute

  onClick = "alert('Mouse click!');"

  onClick = "myHandler(); "

  onClick = "myHandler(42);"

- 2. By assigning the appropriate event property of the DOM object corresponding to the tag to the handler function

  var dom=document.getElementById("myButton");

  dom.onclick = doSomething();

  – A problem with this technique is that no parameters can be passed.

## Events and Event Handling (continued)

- Example: the `load` event - triggered when the loading of a document is completed

```
<!-- load.html
     An example to illustrate the load events
     -->
<html>
<head>
<title> The onLoad event handler>
 </title>
<script type = "text/javascript">
<!--
// The onload event handler

function load_greeting () {
  alert("You are visiting the home page of \n"
         + "Pete's Pickled Peppers \n"
         + "WELCOME!!!");
}
// -->
</script>
</head>

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

See `load.html`

## Events and Event Handling (continued)

- *Radio buttons*

```
 <input type = "radio" name = "button_group"
        value = "blue" onClick = "handler()">
```

- The `checked` property of a radio button object is true if the button is pressed

- Can't use the element's name to identify it, because all buttons in the group have the same name

- Must use the DOM address of the element, e.g.,

```
   var radioElement = document.getElementsById(
"myForm").elements;
```

- Now we have the DOM address of the array of elements of the form

```
   for (var index = 0;
        index < radioElement.length; index++) {
     if (radioElement[index].checked) {
       element = radioElement[index].value;
       break;
     }
   }
```

See radio_click.html & Figures 5.3 & 5.4

# Another way of handling radio buttons

- Alternatively, we can make use of the parameters we can pass to the handlers to simplify our code:
- See radio_click_params.html

## Events and Event Handling (continued)

- *Checking Form Input*

- A good use of JavaScript, because it finds errors in form input before it is sent to the server for processing

Offloads processing of form errors to client

- *Things that must be done*:

1. Detect the error and produce an `alert` message
2. Put the element in focus (the `focus` function)
3. Select the element (the `select` function).

## Events and Event Handling
(continued)

- The `focus` function puts the element in focus, which puts the cursor in the element

```
document.getElementById("phone").focus();
```

- The `select` function highlights the entered (but faulty) value so that when the user enters a new value, the old one is automatically erased first.

- Neither `select` nor `focus` work with NS 6.2, but do work with 7.0 and above
- If event handler returns `false`, the browser will not perform default actions of that event. This is especially important for the submit event: the handler should usually check for proper form completion, and return `false` if all is not well. Consequently, the browser will *not* submit the form data to the server.

## *Example* – comparing passwords

- If a password will be used later, the user is asked to type it in twice
- The program must verify that the second typing of the password is the same as the first
- The form has 4 elements: 2 password input boxes and a Reset and Submit button
- The event handler is triggered by the Submit button
- *Handler actions*:
  1. If no password has been typed in the first box, focus on that box and return false
  2. If the two passwords are not the same, focus and select the first box and return false, else return true
- --> See: pswd_chk.html & Figures 5.5 & 5.6

## Events and Event Handling

- *Another Example* – Checking the format of a name and phone number

- The event handler will be triggered by the `change` event of the text boxes for the name and phone number

- If an error is found in either, an `alert` message is produced and both focus and select are called on the text box element

- Another event handler is used to produce a thank you `alert` message when the input is ok

→ SHOW `validator.html` & Figures 5.7 & 5.8

## The DOM 2 Event Model

- Does not include all DOM 0 features, but they are still supported
- Much more powerful than the DOM 0 model. Analogous to Java's event-handling
- Microsoft does not support it, yet
- Event propagation
  - The node of the document tree where the event is created is called the *target node* (like a Java event source)
  - The first phase is called the *capturing phase*
  - Events begin at the root and move toward the target node:
    - » If there are registered, *enabled*, event handlers at nodes along the way (before the target node is reached), they are run
  - The second phase is at the target node. If there are registered handlers there for the event, they are run
  - The third phase is the *bubbling phase*, like exception handling.
    - » Event goes back to the root; all encountered registered, non-enabled, handlers are run

## The DOM 2 Event Model
(continued)

- Not all events bubble (e.g: load & unload)

- Any handler can stop further propagation by calling the `stopPropagation` method of the `Event` object

- DOM2 model uses the `Event` object method, `preventDefault` to stop default operations, such as submission of a form, when an error has been detected

- Event handler registration is done with the addEventListener method

- Three parameters:

1. Name of the event, as a string literal
2. The handler function
3. A Boolean value that specifies whether the event is enabled during the capturing phase

```
node.addEventListener("change", chkName,
false);
```

## The DOM 2 Event Model

(continued)

- A temporary handler can be created by registering it and then unregistering it with remove `EventListener`

- The `currentTarget` property of `Event` always references the object on which the handler is being executed, while `target` refers to the event source.

- The MouseEvent object (a subobject of Event) has two properties, clientX and clientY, that have the x and y coordinates of the mouse cursor, relative to the upper left corner of the browser window

- An example: A revision of validator, using the DOM 2 event model

→ SEE: `validator2.html`

- Note: DOM 0 and DOM 2 event handling can be mixed in a document

## The `navigator` object

- Indicates which browser is being used

- Two useful properties

1. The `appName` property has the browser's name

2. The `appVersion` property has the version #

- Microsoft has chosen to set the `appVersion` of IE6 to 4 (?)

- Netscape has chosen to set the `appVersion` of NS6 to 5.0 (?)

→ SHOW `navigator.html` & Figures 5.9 & 5.10