Events Handling using Java script

Events and Event Handling

- *Event-driven programming* is a style of programming in which pieces of code, *event handlers*, are written to be activated when certain *events* occur
- Events represent activity in the environment including, especially, user actions such as moving the mouse or typing on the keyboard
- An *event handler* is a program segment designed to execute when a certain event occurs
- Events are represented by JavaScript objects
- Assign an event attribute an event handler

Events, Attributes

Event	*Tag Attribute*	
blur	onblur	
change	onchange	
focus	onfocus	
load	onload	Window events
unload	onunload	
click	onclick	
mousedown	onmousedown	Button events
mouseup	onmouseup	
mousemove	onmousemove	
mouseout	onmouseout	
mouseover	onmouseover	Link events
select	onselect	
submit	onsubmit	

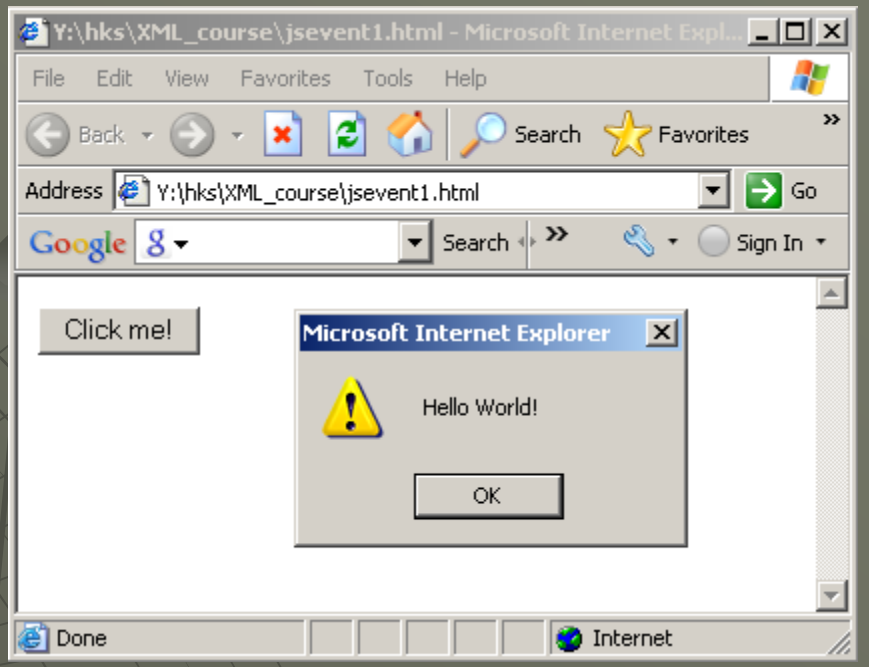
Events, Attributes and Tags

- Particular events are associated to certain attributes
- The attribute for one kind of event may appear on different tags allowing the program to react to events affecting different components
- 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
- Losing the focus is *blur*ring
- Javascript events can be interactive (mouse click) or non interactive (page load)

Simple button event Example

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
    alert("Hello World!");
}
</script>
</head>
<body>
<form>
<input type="button" value="Click me!"
   onclick="displaymessage()" >
</form>
</body>

</html>
```

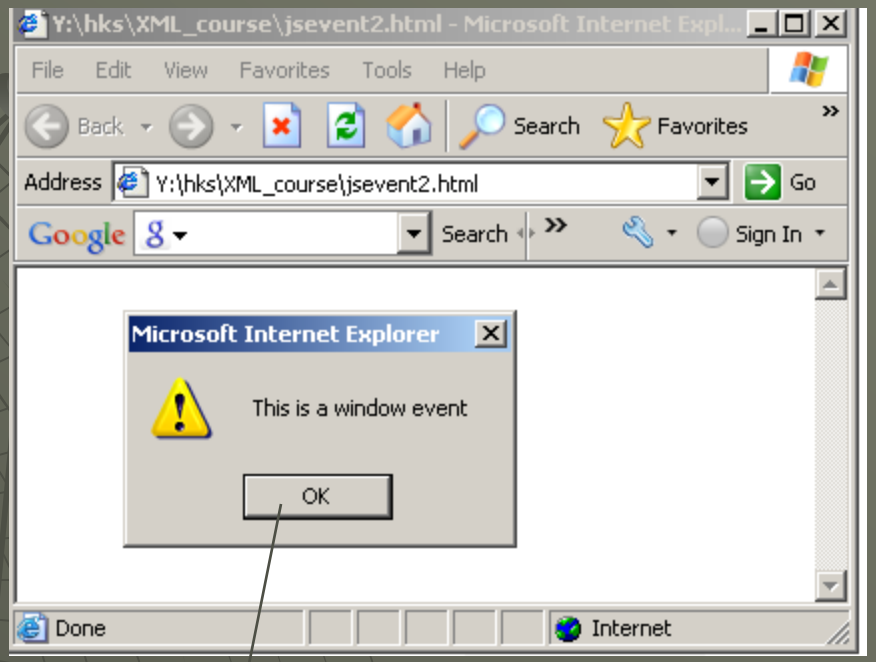


Execute Ex

Handling Events from Body Elements Simple window event Example

```
<html>
<head>
<script type="text/javascript">
function load()
{
   window.alert("Page is loaded”);
   window.status=“This is a window event”;
}
</script>
</head>
<body onload="load()">
</body>
</html>
```

window.alert



window.status

Note : This example illustrates a script that is run when the page first loads

Execute Ex

Simple Link Event Example

- The onmouseover event occurs when the mouse pointer moves over a specified object or a link.
- Demo Example
- Source Code

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
- The onfocus event occurs when an object gets focus.
- The onblur event occurs when an object loses focus.

onfocus() & onblur() example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
   lang="en">
<head>
<title>XHTML onfocus and onblur Example</title>
</head>
<body>
<form>
<input type="text" onfocus="javascript:alert('text box is focussed')"
            onblur="javascript:alert('text box not focussed')">
</form>
</body>
</html>
```

Demo

password field in text box

```
<html>
<body>

<form>
Username:
<input type="text" name="user">
<br>
Password:
<input type="password" name="password">
</form>
<p>
Note that when you type characters in a password field,
the browser displays asterisks or bullets instead of the
    characters.
</p>
</body>
</html>
```

Validating Form Input using getElementById()

- **We can get the elements in a form using getElementById().**
- **The getElementById() method returns a reference to the first object with the specified ID**
- **Syntax : document.getElementById(param1)**
  Parameters: param1 Required; the element's id value

```
<html>
<head>
<title> GetElementID ex</title>
</head>
<body>
<script type="text/javascript">
function notEmpty()
{
var myTextField = document.getElementById('myText');
if(myTextField.value != "")
   alert("You entered: " + myTextField.value)
else
alert("Would you please enter some text?")
}
</script>
<form>
<input type='text' id='myText' />
<input type='button' onclick='notEmpty()'
  value='Form Checker' />
</form>
</body>
</html>
```

Demo

Exercise (1 hour)

- Develop, test, and validate an XHTML document that collects the following information from the user: last name, first name, middle initial, age (restricted to be greater than 17, and weight (restricted to the range of 45-75). You must have event handlers for the form elements that collect this information that check the input data for correctness. Messages in *alert* windows must be produced when errors are detected.

Object Creation and Modification

- The new expression is used to create an object
  - This includes a call to a *constructor*
  - The new operator creates a blank object, the constructor creates and initializes all properties of the object
- Properties of an object are accessed using a dot notation: *object*.*property*
- Properties are not variables, so they are not declared
  - An object may be thought of as a Map/Dictionary/Associative-Storage
- The number of properties of an object may vary dynamically in JavaScript

Dynamic Properties

- Create my_car and add some properties

```
// Create an Object object
var my_car = new Object();
// Create and initialize the make
  property
my_car.make = "Ford";
// Create and initialize model
my_car.model = "Contour SVT";
```

- The delete operator can be used to delete a property from an object
  - `delete my_car.model`

The for-in Loop

- Syntax

  for (*identifier* in *object*)

  statement or compound statement
- The loop lets the identifier take on each property in turn in the object
- Printing the properties in my_car:

```
for (var prop in my_car)
  document.write("Name: ", prop, "; Value: ",
   my_car[prop], "<br />");
```

- Result:

```
Name: make; Value: Ford
Name: model; Value: Contour SVT
```

Arrays

- Arrays are lists of elements indexed by a numerical value
- Array indexes in JavaScript begin at 0
- Arrays can be modified in size even after they have been created

`Array` Object Creation

- Arrays can be created using the new Array method
  - new Array with one parameter creates an empty array of the specified number of elements
    - new Array(10)
  - new Array with two or more parameters creates an array with the specified parameters as elements
    - new Array(10, 20)
- Literal arrays can be specified using square brackets to include a list of elements
  - var alist = [1, “ii”, “gamma”, “4”];
- Elements of an array do not have to be of the same type

Characteristics of `Array` Objects

- The length of an array is one more than the highest index to which a value has been assigned or the initial size (using Array with one argument), whichever is larger
- Assignment to an index greater than or equal to the current length simply increases the length of the array
- Only assigned elements of an array occupy space
  - Suppose an array were created using new Array(200)
  - Suppose only elements 150 through 174 were assigned values
  - Only the 25 assigned elements would be allocated storage, the other 175 would not be allocated storage

`Array` Methods

- join
- reverse
- sort
- concat
- slice

Dynamic List Operations

- push
  - Add to the end
- pop
  - Remove from the end
- shift
  - Remove from the front
- unshift
  - Add to the front

Using Regular Expressions

- Regular expressions are used to specify patterns in strings
- JavaScript provides two methods to use regular expressions in pattern matching
  - String methods
  - RegExp objects (not covered)
- A literal regular expression pattern is indicated by enclosing the pattern in slashes
- The search method returns the position of a match, if found, or -1 if no match was found

Example Using search

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position > 0)
       document.write("'bits' appears in position",
              position, "<br />");
else
       document.write(
              "'bits' does not appear in str <br />");
```

- This uses a pattern that matches the string ‘bits’
- The output of this code is as follows:

  'bits' appears in position 3

Characters and Character-Classes

- *Metacharacters* have special meaning in regular expressions
  - `\ | ( ) [ ] { } ^ $ * + ? .`
  - These characters may be used literally by escaping them with \
- Other characters represent themselves
- A period matches any single character
  - /f.r/ matches for and far and fir but not fr
- A character class matches one of a specified set of characters
  - [*character set*]
  - List characters individually: [abcdef]
  - Give a range of characters: [a-z]
  - Beware of [A-z]
  - ^ at the beginning negates the class

Predefined character classes

Name	Equivalent Pattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character (alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A whitespace character
\S	[^ \r\t\n\f]	Not a whitespace character

Repeated Matches

- A pattern can be repeated a fixed number of times by following it with a pair of curly braces enclosing a count
- A pattern can be repeated by following it with one of the following special characters
  - * indicates zero or more repetitions of the previous pattern
  - + indicates one or more of the previous pattern
  - ? indicates zero or one of the previous pattern
- Examples
  - `/\(\d{3}\)\d{3}-\d{4}/` might represent a telephone number
  - `/[$_a-zA-Z][$_a-zA-Z0-9]*/` matches identifiers

Anchors

- Anchors in regular expressions match positions rather than characters
  - Anchors are 0 width and may not take multiplicity modifiers
- Anchoring to the end of a string
  - ^ at the beginning of a pattern matches the beginning of a string
  - $ at the end of a pattern matches the end of a string
    - The $ in /a$b/ matches a $ character
- Anchoring at a word boundary
  - \b matches the position between a word character and a non-word character or the beginning or the end of a string
  - /\bthe\b/ will match ‘the’ but not ‘theatre’ and will also match ‘the’ in the string ‘one of the best’