

Advanced JavaScript Programming

The HTML Document Object Model

Lesson 1, Activity 2: `innerHTML`

Most HTML elements have an `innerHTML` property, which can be used to access and modify the HTML within that element. The `innerHTML` property wasn't included in any specification until HTML5. However, as it's extremely useful and well supported, we will use it widely throughout this course.

Lesson 1, Activity 3: Accessing Element Nodes

innerHTML

Most HTML elements have an `innerHTML` property, which can be used to access and modify the HTML within that element. The `innerHTML` property wasn't included in any specification until HTML5. However, as it's extremely useful and well supported, we will use it widely throughout this course.

innerHTML Illustration:

Given the code:

```
<p>I <strong>love</strong> Webucator.</p>
```

the `innerHTML` property of the `<p>` tag would be: "I love Webucator."

Tip: you can use the `innerHTML` property to either **get** the element's inner HTML value (as shown above) or you can use it to **set** the element's inner HTML value. More on this later in the lesson.

JavaScript provides several different ways to access elements on the page. Let's have a look.

The `document.getElementById()` Method

Chances are you have already seen the `document.getElementById()` method, which returns the first element with the given `id` (there shouldn't be more than one on the page!) or `null` if none is found. The following example illustrates how `getElementById()` works:

Code Sample:

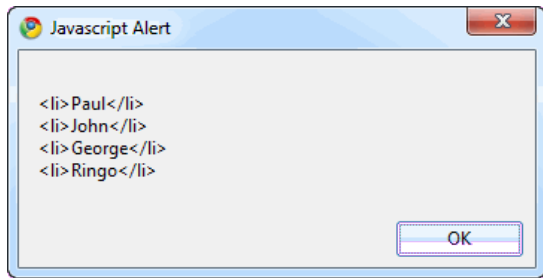
HTMLDOM/Demos/getElementById.html

```
---- C O D E   O M I T T E D ----

function show() {
    var elem = document.getElementById("BeatleList");
    alert(elem.innerHTML);
}
---- C O D E   O M I T T E D ----

<body onload="show();">
<h1>Rockbands</h1>
<h2>Beatles</h2>
<ol id="BeatleList">
  <li>Paul</li>
  <li>John</li>
  <li>George</li>
  <li>Ringo</li>
</ol>
<h2>Rolling Stones</h2>
<ol id="StonesList">
  <li>Mick</li>
  <li>Keith</li>
  <li>Charlie</li>
  <li>Bill</li>
</ol>
</body>
</html>
```

When this page loads, the following alert box will pop up:



The `element.getElementsByTagName()` Method

The `element.getElementsByTagName()` method of an element node retrieves all descendant (children, grandchildren, etc.) elements of the specified tag name and stores them in a `NodeList`, which is similar, but not exactly the same as an array of nodes. The following example illustrates how `element.getElementsByTagName()` works:

Code Sample:

HTMLDOM/Demos/getElementsByTagName.html

```

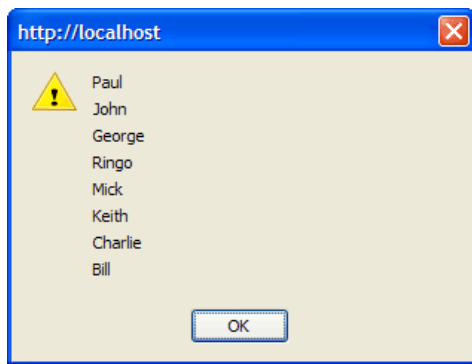
---- C O D E   O M I T T E D ----

function getElements()
{
  var elems = document.getElementsByTagName("li");
  var msg="";
  for (var i=0; i < elems.length; i++)
  {
    msg += elems[i].innerHTML + "\n";
  }
  alert(msg);
}
</script>
<title>getElementsByTagName()</title>
</head>

<body onload="getElements();">
<h1>Rockbands</h1>
<h2>Beatles</h2>
<ol>
  <li>Paul</li>
  <li>John</li>
  <li>George</li>
  <li>Ringo</li>
</ol>
<h2>Rolling Stones</h2>
<ol>
  <li>Mick</li>
  <li>Keith</li>
  <li>Charlie</li>
  <li>Bill</li>
</ol>
</body>
</html>

```

When this page loads, the following alert box will pop up:



As you can see, the method returns all the node's descendants, not just the direct children.

The `document.getElementsByClassName()` Method

The `document.getElementsByClassName()` method is well supported by browsers (with one major exception) and is officially part of HTML5. Applicable to all elements that can have descendant elements, `document.getElementsByClassName()` is used to retrieve all the descendant (children, grandchildren, etc.) elements of a specific class. For example, the following code would return a node list containing all elements of the "warning" class:

```
var warnings = document.getElementsByClassName("warning");
```

Unfortunately, `document.getElementsByClassName()` is not supported by IE until version 9, but we can use this helper function for backward compatibility:

```
function getElementsByClassName(node, classSearch) {
  if (node.getElementsByClassName) {
    //returns nodeList
    return node.getElementsByClassName(classSearch);
  } else {
    //returns array
    var classElements = [];
    var elems = node.getElementsByTagName("*");
    var pattern = new RegExp("(^|\\s)" + classSearch + "(\\s|$)");
    for (var i in elems) {
      if (pattern.test(elems[i].className) ) {
        classElements.push(elems[i]);
      }
    }
    return classElements;
  }
}
```

This function, included in [ClassFiles/lib.js](#), will use the browser's native `document.getElementsByClassName()` if it exists. Otherwise, it traverses the `nodeTree` finding elements with the specified class name and returns an array of matched elements.

The `document.querySelectorAll()` Method

The `document.querySelectorAll()` method gives you access to elements using the CSS selector syntax. For example, the following code would return a node list containing all list items that are direct children of `ol` tags:

```
var orderedListItems = document.querySelectorAll("ol>li");
```

The `document.querySelector()` Method

The `document.querySelector()` method is the same as `document.querySelectorAll()` but rather than returning a node list, it returns only the first element found. The following two lines of code would both return the first list item found in an `ol` tag:

```
var firstOrderedListItem = document.querySelectorAll("ol>li")[0];
var firstOrderedListItem = document.querySelector("ol>li");
```

The `querySelectorAll()` and `querySelector()` methods are not supported until version 8 of Internet Explorer and version 3.5 of Firefox and they are, unfortunately, difficult to reproduce. Note that popular JavaScript libraries such as [YUI](#) and [jQuery](#) implement cross-browser CSS selector utilities.

Accessing Element and Text Nodes Hierarchically

There are several node methods and properties that provide access to other nodes based on their hierarchical relationship. The most common and best supported of these are shown in the table below.

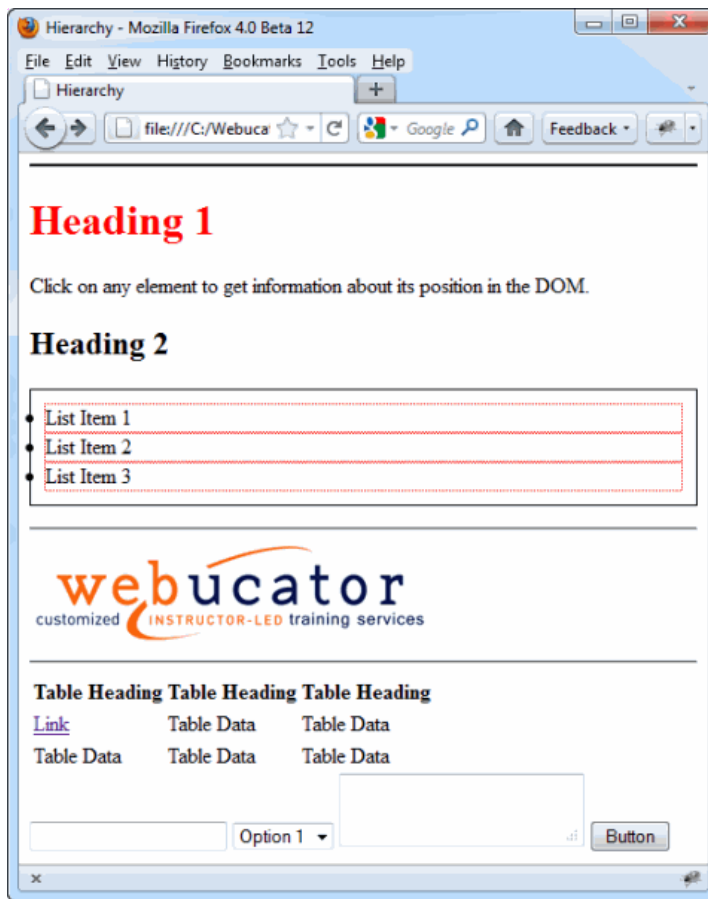
Properties for Accessing Element Nodes

Property	Description
<code>childNodes[]</code>	A <code>nodeList</code> containing all of a node's child nodes.
<code>firstChild</code>	A reference to a node's first child node.
<code>lastChild</code>	A reference to a node's last child node
<code>nextSibling</code>	A reference to the next node at the same level in the document tree.
<code>parentNode</code>	A reference to a node's parent node.
<code>previousSibling</code>	A reference to the previous node at the same level in the document tree.

The `this` Object

The `this` keyword provides access to the current object. It references the object in which it appears.

The following example shows a page ([HTMLDOM/Demos/Hierarchy.html](#)) on which a mouse click on any element will display a message giving information about that element's position in the object hierarchy.



When you click on the middle list item, for example, you get the following message:

You clicked on a LI element.

1. parentNode: UL
2. firstChild: #text
3. lastChild: #text
4. nextSibling: #text
5. previousSibling: #text
6. childNodes.length: 1

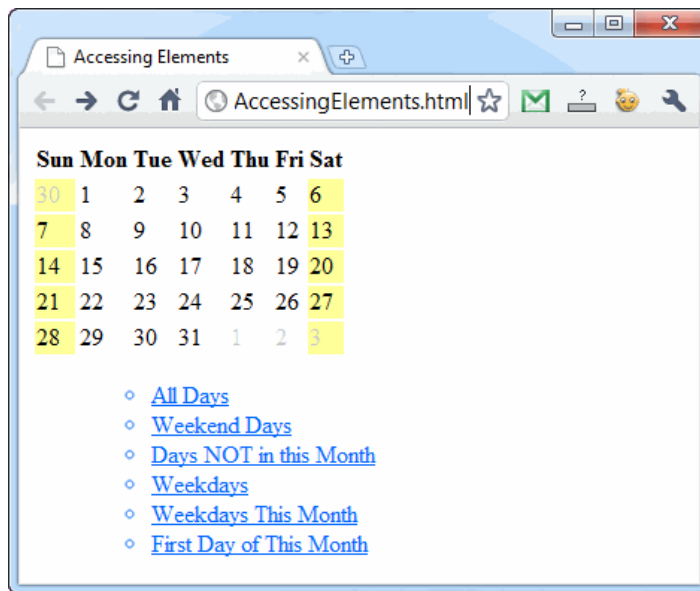
This shows that the list item's parent is an unordered list (UL), its first child is a text node ("List Item 2"), its last child is the same as it only has one child, its next and previous siblings are both text nodes, and it has only one child node (the text node). Again, this message was generated in Firefox, but would be the same in other modern browsers. However, as we'll see later, the result will be slightly different in Internet Explorer 8 and earlier.

Lesson 1, Activity 5: Accessing Elements

Duration: 25 to 40 minutes.

In this exercise, you will practice using the methods for accessing elements.

1. Open HTMLDOM/Solutions/AccessingElements.html in your **browser**. It will look like this:



2. Click on the links and notice how the calendar changes. For example, click on **Weekdays This Month** and the calendar will appear as follows:

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

3. Now open HTMLDOM/Exercises/AccessingElements.html in your **editor**.
4. Notice that [ClassFiles/lib.js](#) is included and that there are four functions already written:
 1. `highlight()` - adds the "highlight" class to the passed-in node or nodes.
 2. `unhighlight()` - removes the "highlight" class from the passed-in node or nodes.
 3. `unhighlightDays()` - helper function removing the "highlight" class from all the `td` elements to reset the calendar.
 4. `weekEndDays()` - gets all the weekend days using the `getElementsByClassName()` helper function from the [ClassFiles/lib.js](#) library and passes it to the `highlight()` function after calling `unhighlightDays()` to reset the calendar.
5. Complete the `offDays()` function to highlight the days that are not in the current month (that have the "off" class).
6. Complete the `allDays()` function to highlight all the days of the month.
7. Complete the `firstDayOfMonth()` function to highlight the first day of the current month (the one that contains a value of "1").
8. Complete the `weekDays()` function to highlight all the week days. Use `querySelectorAll` if the browser supports it.
9. **Challenge:** Complete the `weekDaysThisMonth()` function to highlight all the weekdays of the month. **Hint:** You can make use of the `weekDays()` function.

The `weekDays()` and `weekDaysThisMonth()` functions will break in IE8 because it does support the `querySelectorAll()` method, but does not support the specific CSS selectors passed to it. One way to handle this would be to use a try/catch block.

Solution:

HTMLDOM/Solutions/AccessingElements.html


```

---- C O D E   O M I T T E D ----
function offDays() {
    var calendar = document.getElementById("calendar");
    var offDays = getElementsByClassName(calendar,"off");
    unhighlightDays();
    highlight(offDays);
    return offDays;
}

function allDays() {
    var calendar = document.getElementById("calendar");
    var days = calendar.getElementsByTagName("td");
    highlight(days);
    return days;
}

function firstDayOfMonth() {
    var calendar = document.getElementById("calendar");
    var headerRow = calendar.rows[0];
    var ths = headerRow.getElementsByTagName("th");
    var firstRow = calendar.rows[1];
    var tds = firstRow.getElementsByTagName("td");
    unhighlightDays();
    for (var i=0; i<tds.length; i++) {
        if ( tds[i].innerHTML==1 ) {
            highlight(tds[i]);
            return tds[i];
        }
    }
}

function weekDays() {
    var weekDays;
    if ( document.querySelectorAll ) {
        weekDays = document.querySelectorAll("td:not(.weekend)");
    } else {
        var calendar = document.getElementById("calendar");
        var days = calendar.getElementsByTagName("td");
        var weekDays = [];
        for (var i=0; i<days.length; i++) {
            if ( !hasClassName(days[i],"weekend") ) {
                weekDays.push(days[i]);
            }
        }
    }
    unhighlightDays();
    highlight(weekDays);
    return weekDays;
}

function weekDaysThisMonth() {
    var weekDaysThisMonth=[];
    var days=weekDays();
    var i;
    for (i=0; i<days.length; ++i) {
        if ( !hasClassName(days[i],"off") ) {
            weekDaysThisMonth.push(days[i]);
        }
    }
    unhighlightDays();
    highlight(weekDaysThisMonth);
    return weekDaysThisMonth;
}
---- C O D E   O M I T T E D ----

```

Lesson 1, Activity 6: Attaching Events

It is possible to attach all sorts of events, such as clicks, mouseovers, mouseouts, etc., to elements in the DOM. The simplest, standard cross-browser method for adding events is to assign an event handler property to a node. The syntax is shown below:

```
node.onevent = DoSomething;
```

However, there is a better way to go about this. In [ClassFiles/lib.js](#), there is a cross-browser `observeEvent()` function shown below:

```
/*
Function Name: observeEvent
Arguments: target, eventName, observerFunction, useCapture
Action: cross browser: attaches eventName to passed in target. observerFunction will be the event's callback function.
*/
function observeEvent(target, eventName, observerFunction, useCapture){
  if (target.addEventListener) {
    target.addEventListener(eventName, observerFunction, useCapture);
  } else if (target.attachEvent) {
    target.attachEvent("on" + eventName, observerFunction);
  }
}
```

The `addEventListener()` node method is part of the DOM specification. It is supported by most modern browsers, but not by older versions of Internet Explorer. In IE8 and earlier, you must use IE's proprietary `attachEvent()` method, which essentially works the same way. The only difference is that in IE, the `attachEvent()` method will repeatedly attach the same event to an event handler when called multiple times using the same parameters. Our `observeEvent()` function above attaches events to nodes in a cross-browser fashion. Throughout the course, you'll find this in the [ClassFiles/lib.js](#) library.

Don't worry about the `useCapture` argument just yet. We'll explain it soon.

This `observeEvent()` function has a big advantage over the `node.onevent` style: it allows for multiple callback functions to be attached to a single event. The following example illustrates this.

Code Sample:

[HTMLDOM/Demos/Events.html](#)

```
--- C O D E   O M I T T E D ---

<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
function setEvents1() {
  var trigger = document.getElementById("trigger");
  trigger.onclick=sayHi;
  trigger.onclick=sayBye;
}

function setEvents2() {
  var trigger = document.getElementById("trigger");
  observeEvent(trigger,"click",sayHi);
  observeEvent(trigger,"click",sayBye);
}

function sayHi() {
  alert("Hi!");
}

function sayBye() {
  alert("Bye!");
}
```

```
observeEvent(window,"load",function() {
  observeEvent(button1,"click",setEvents1);
  observeEvent(button2,"click",setEvents2);
});
</script>
---- C O D E   O M I T T E D ----
```

When the page loads, the "trigger" button has no events attached to it.

1. Click on the button that reads "Attach events using `node.onevent` syntax," which runs the `setEvents1()` function. Then click on the "trigger" button. A single "Bye" alert pops up. This is because the call to `sayHi()` was replaced by the call to `sayBye()`.
2. Refresh the page and then click on the button that reads "Attach events using `observeEvent()` function," which runs the `setEvents2()` function. Now click on the "trigger" button again. Both the "Hi" and "Bye" alerts pop up. The order in which they pop up may depend on which browser you are using.

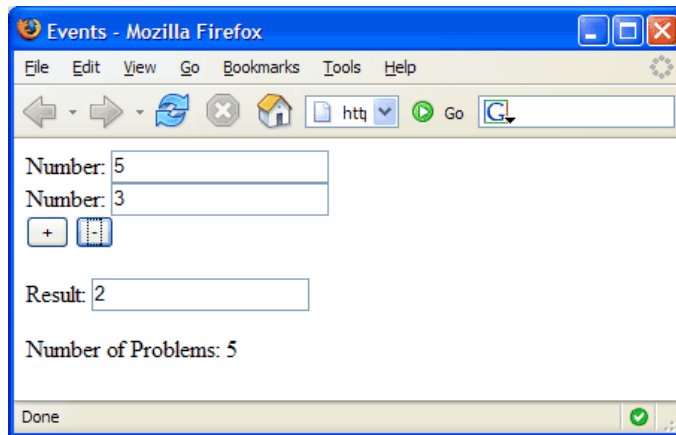
Being able to have multiple callback functions associated with a single object event can be useful. As an example, you might have a `calculation()` function that is triggered when a button is clicked. You may later want to make that same action (the click of the button) cause another part of the page to be updated. As these are two separate tasks, you would separate them into different functions, both of which can be tied to the same event.

Lesson 1, Activity 8: Attaching Events

Duration: 10 to 20 minutes.

In this exercise you will use the `observeEvent()` function to attach multiple callback functions to the same event.

1. Open HTMLEDOM/Exercises/AttachEvent.html for editing. A screen shot of the page is shown below:



2. In the `init()` function, write code so that...
 - when `addButton` is clicked the `addNumbers()` function is called.
 - when `subtractButton` is clicked the `subtractNumbers()` function is called.
 - when either button is clicked the `incrementProblems()` function is called.
3. To test your solution, open HTMLEDOM/Exercises/AttachEvent.html in your browser.
 - When you click on the plus button, it should add the two numbers and increment the number of problems shown by one.
 - When you click on the minus button, it should subtract the number 1 from number 2 and increment the number of problems shown by one.

Code Sample:

HTMLEDOM/Exercises/AttachEvent.html

```

---- CODE OMITTED ----

observeEvent(window,"load",init);
function init() {
  var addButton = document.getElementById("addButton");
  var subtractButton = document.getElementById("subtractButton");

  /*
  Write code so that...
  - when addButton is clicked the addNumbers() function is called
  - when subtractButton is clicked the subtractNumbers() function is called
  - when either button is clicked the incrementProblems() function is called
  */
}

function addNumbers(e) {
  e = e || window.event;
  var target = e.target || e.srcElement;
  var num1 = target.form.num1.value;
  var num2 = target.form.num2.value;
  var result = Number(num1) + Number(num2);
  target.form.result.value = result;
}

function subtractNumbers(e) {
  e = e || window.event;
  var target = e.target || e.srcElement;
  var num1 = target.form.num1.value;
  var num2 = target.form.num2.value;
  var result = num1 - num2;
  target.form.result.value = result;
}

```

```

}

var numProbs=0;
function incrementProblems() {
    numProbs++;
    document.getElementById("numProblems").innerHTML = numProbs;
}
---- C O D E   O M I T T E D ----

```

Solution:

HTMLDOM/Solutions/AttachEvent.html

```

---- C O D E   O M I T T E D ----

function init() {
    var addButton = document.getElementById("addButton");
    var subtractButton = document.getElementById("subtractButton");
    observeEvent(addButton,"click",addNumbers);
    observeEvent(subtractButton,"click",subtractNumbers);
    observeEvent(addButton,"click",incrementProblems);
    observeEvent(subtractButton,"click",incrementProblems);
}
---- C O D E   O M I T T E D ----

```

Lesson 1, Activity 9: Event Propagation: Capturing and Bubbling

Let's look back at the `useCapture` argument in the `observeEvent()` function. Consider a user who clicks on a word in a table cell. The user's intent might be to click the word, the cell, the row, or maybe even the entire table. To illustrate, take a look at the following code sample.

Code Sample:

HTMLDOM/Demos/event-propagation-table.html

```
---- CODE OMITTED ----
---- CODE OMITTED ----
```

The Beatles

John	Lennon	
Paul	McCartney	
Ringo	Starr	
George	Harrison	

---- CODE OMITTED ----

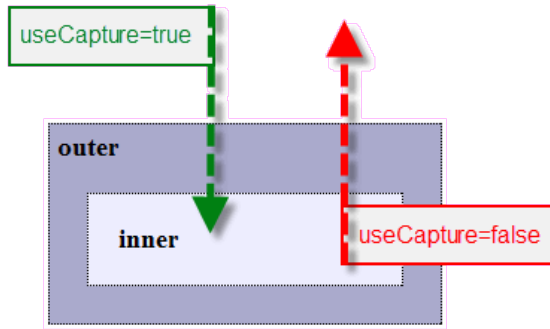
Notice that both the `trs` and the `imgs` have `onclick` event handlers:

- When the user clicks on any of the row itself, an alert pops up stating the birthday of that Beatle. (Imagine a more informative detail window.)
- When the user clicks on the **edit** icon, an alert pops up reading "Edit record form: 1," where 1 is the id of the row. (Imagine a real edit form.) In this case, we don't want to show the "birthday" alert. But it does pop up.

We will look at how to fix this shortly, but first, let's consider the order of the alerts.

In IE8 and earlier, events are captured starting with the innermost element, in this case the image, and then "bubble up" to the parent elements. There is no easy way to change this behavior. So, using IE8 and earlier, when you click on the **edit** icon in our demo, the "edit form" alert will pop up first followed by the "birthday" alert.

In all other modern browsers, you have a choice between "bubbling up" or "trickling down". The trickling down phase is called the *capture* phase. The third argument of the built-in `addEventListener()` method is `useCapture`. When `true`, it uses the "trickle down" behavior. When `false`, it "bubbles up" as illustrated in the diagram below:



Our `observeEvent()` function also takes a `useCapture` argument, which it passes into the built-in `addEventListener()` method. If we don't pass in a value for `useCapture`, the value will be *undefined*, which is *falsey*. This will make modern browsers work in the same way as IE8 and earlier (bubbling up).

This behavior can be a bit complicated to understand, so before moving on, let's take a look at the following two examples:

Code Sample:

HTMLDOM/Demos/bubble-up.html

```

---- C O D E   O M I T T E D ----

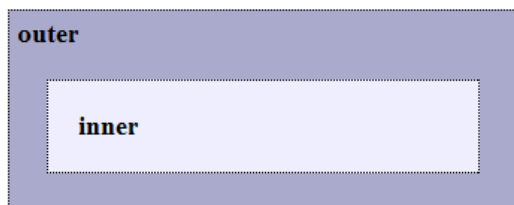
Event Propagation: Bubbling Up

Bubbling Up
outer
inner

```

Click on the inner `div` and you'll see this:

Bubbling Up



1. inner `div` clicked
2. outer `div` clicked

Code Sample:

HTMLDOM/Demos/trickle-down.html

```

---- C O D E   O M I T T E D ----

Event Propagation: Trickling Down

```

```
Trickling Down
outer
  inner
```

Click on the inner `div` and you'll see this:

Trickling Down



1. outer div clicked
2. inner div clicked

Notice the different order of the list. The only difference between these two files is the value of the `useCapture` argument.

Back to our earlier table example. In that example, we passed in `false` for `useCapture` so that the behavior across browsers would be consistent. Now, we need to stop that second alert from popping up. Most modern browsers stop events from propagating using the `stopPropagation()` method of the event, which automatically gets passed into a *callback* function. Look at the following example, which is a modification of HTMLDOM/Demos/bubble-up.html:

Code Sample:

HTMLDOM/Demos/stop-prop.html

```
---- C O D E   O M I T T E D ----

observeEvent(window,"load",function() {
  var output = document.getElementById("output");
  observeEvent(document.getElementById("outer"),"click",function() {
    output.innerHTML+=

      • outer div clicked
    ";
  }, false);
  observeEvent(document.getElementById("inner"),"click",function(e) {
    output.innerHTML+=

      • inner div clicked
    ";
    e.stopPropagation();
  }, false);
});
---- C O D E   O M I T T E D ----
```

The two things to notice:

1. The anonymous function passed into the `observeEvent()` function on line 8 now takes an argument: `e`.
2. After writing out the "inner div clicked" list item, we call `e.stopPropagation()` to put a stop to the bubbling up.

The result is that the second line item ("outer div clicked") doesn't appear.

And it would be as easy as that, except that this won't work in IE until version 9 for two reasons:

1. In IE8 and earlier, the event was not automatically passed in to the callBack function. Instead, the last event was stored in `window.event`.
2. IE8 and earlier provided a different way of stopping event propagation. Instead of calling `stopPropagation()` you have to set the `cancelBubble` property of the event to true: `e.cancelBubble = true;`.

In [ClassFiles/lib.js](#), there is a cross-browser `stopPropagation()` function (shown below) that addresses the browser differences. You can use this safely to stop events from propagating.

```
function stopPropagation(e) {
  e = e || window.event;
  if (e.stopPropagation) {
    e.stopPropagation();
  } else {
    e.cancelBubble = true;
  }
}
```

So we can finally return to our Beatles table and show the working code:

Code Sample:

[HTMLDOM/Demos/event-propagation-table-fixed.html](#)

```
---- C O D E   O M I T T E D ----

---- C O D E   O M I T T E D ----
```

The call to `stopPropagation()` will stop the event from bubbling up to the table row, so we'll just get the "edit form" alert.

Lesson 1, Activity 10: A Simple Soccer Game

Duration: 15 to 25 minutes.

In this exercise, you will finish creating a very simple game, in which the user must click a soccer ball before it moves off the field. The field looks like this:

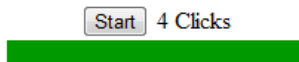
Click the Ball



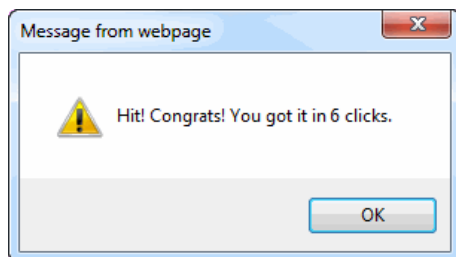
1. Open HTMLDOM/Exercises/soccer-game.html in your editor and study the code. Don't worry about the `moveBall()` function. We'll learn more about how to do that kind of thing when we look at Dynamic HTML.
2. Notice that when the page loads, we attach an event to the **Start** button that triggers the `start()` function.
3. Write code that triggers the `miss()` function when the user **clicks down and up** on the field and triggers the `hit()` function when the user **clicks down** on the ball.
4. Make sure that when the user clicks the ball, the `miss()` function doesn't execute.

Challenge

1. Add code so that the number of clicks displays next to the **Start** button:



2. Add code to the `hit()` function so that the number of attempts it took to get a hit is displayed in the alert message:



Code Sample:

HTMLDOM/Exercises/soccer-game.html

```

---- CODE OMITTED ----

var timer = null;
observeEvent(window,"load",function() {
  observeEvent(document.getElementById("start"),"click",start,false);
});

function miss() {
  alert("Miss!");
}

function hit(e) {
  alert("Hit! Congrats!");
}

```

```

clearInterval(timer);
}

function start() {
  var ball = document.getElementById("ball");
  ball.style.top="92px";
  ball.style.left="92px";
  timer=setInterval(function() { moveBall(ball); },20);
}

function moveBall(ball) {
---- C O D E   O M I T T E D ----

}

---- C O D E   O M I T T E D ----

<h1>Click the Ball</h1>
<div>
  <button id="start">Start</button>
</div>
<div id="field"></div>

---- C O D E   O M I T T E D ----

```

Solution:**HTMLDOM/Solutions/soccer-game.html**

```

---- C O D E   O M I T T E D ----

observeEvent(window,"load",function() {
  observeEvent(document.getElementById("field"),"click",miss,false);
  observeEvent(document.getElementById("ball"),"mousedown",hit,false);
  observeEvent(document.getElementById("start"),"click",start,false);
});
---- C O D E   O M I T T E D ----

```

Challenge Solution:**HTMLDOM/Solutions/soccer-game-challenge.html**

```

---- C O D E   O M I T T E D ----
function miss() {
  incrementClickCount();
  alert("Miss!");
}

function hit(e) {
  incrementClickCount();
  alert("Hit! Congrats! You got it in " + document.getElementById("click-count").innerHTML + " clicks.");
  stopPropagation(e);
  clearInterval(timer);
}
---- C O D E   O M I T T E D ----

function incrementClickCount() {
  var clickCount = document.getElementById("click-count").innerHTML;
  clickCount++;
  document.getElementById("click-count").innerHTML = clickCount;
}
</script>
<title>Click the Ball</title>
</head>
<body>

<h1>Click the Ball</h1>
<div>
  <button id="start">Start</button>

```

```
<span id="click-count">0</span> Clicks  
</div>  
<div id="field"></div>  
</body>  
</html>
```

Lesson 1, Activity 12: Detaching Events

Our game, though extremely advanced and likely to sell well, has one small issue: clicks on the field are captured even before the **Start** button is pushed. Also, the **Start** button should be disabled when the game is active. We need to do the following three things to improve the game:

1. We should not start observing clicks on the field and ball until the **Start** button has been pushed.
2. We should then stop observing those events after the game is finished.
3. We should disable the **Start** button when the game is active.

The first part is easy. We simply move the `observeEvent()` calls for the **Field** and **Ball** to the end of the `start()` function:

```
function start() {
  ... observeEvent(document.getElementById("field"), "click", miss, false);
  observeEvent(document.getElementById("ball"), "mousedown", hit, false)
}
```

To do the second part, we need to learn how to detach events; in other words, to stop observing events. According to the specification, this is done using the `removeEventListener()` method, which takes the same three arguments as the `addEventListener()` method:

```
target.removeEventListener(eventName, observerFunction, useCapture);
```

So, to stop capturing clicks on the **Ball**, we could use this code:

```
document.getElementById("ball").removeEventListener("mousedown", hit, false);
```

Unfortunately, as you might expect, this is handled differently by IE8 and earlier, which uses the following code:

```
target.detachEvent("on" + eventName, observerFunction);
```

We have included the following cross-browser `unObserveEvent()` function in our [ClassFiles/lib.js](#):

```
function unObserveEvent(target, eventName, observerFunction, useCapture){
  if (target.removeEventListener) {
    target.removeEventListener(eventName, observerFunction, useCapture);
  } else if (target.detachEvent) {
    target.detachEvent("on" + eventName, observerFunction);
  }
}
```

Applications with a lot of event listeners can suffer from memory leak issues, so it is important to remove event listeners that are no longer being to used. For a good article on this, see Josh Davis's [JavaScript Built-in Listeners and Memory Leaks](#).

Now back to our soccer game. To stop observing the click events on the **Ball** and **Field**, we can add an `endGame()` function:

```
function endGame() {
  unObserveEvent(document.getElementById("field"), "click", miss, false);
  unObserveEvent(document.getElementById("ball"), "mousedown", hit, false);
}
```

Now the only thing we have left is to disable the **Start** button when the game starts. We can do that by adding this line to the `start()` function:

```
document.getElementById("start").disabled=true;
```

We then have to re-enable it when the game ends, so we add this line to our new `endGame()` function:

```
document.getElementById("start").disabled=false;
```

Here is the complete code with the new bits highlighted:

Code Sample:

[HTMLDOM/Demos/soccer-game-detach-event.html](#)

```
---- C O D E   O M I T T E D ----

var timer = null;
observeEvent(window,"load",function() {
  observeEvent(document.getElementById("start"),"click",start,false);
  //removed observeEvent calls for ball and field
});
---- C O D E   O M I T T E D ----

function start() {
  var ball = document.getElementById("ball");
  document.getElementById("start").disabled=true;
  ball.style.top="92px";
  ball.style.left="92px";
  document.getElementById("click-count").innerHTML="0";
  timer=setInterval(function() { moveBall(ball) },20);

  observeEvent(document.getElementById("field"),"click",miss,false);
  observeEvent(document.getElementById("ball"),"mousedown",hit,false);
}
---- C O D E   O M I T T E D ----

function endGame() {
  unobserveEvent(document.getElementById("field"),"click",miss,false);
  unobserveEvent(document.getElementById("ball"),"mousedown",hit,false);
  document.getElementById("start").disabled=false;
}
---- C O D E   O M I T T E D ----
```

Lesson 1, Activity 14: Accessing Attribute Nodes

The `element.getAttribute()` Method

The `getAttribute()` method of an element node returns the attribute value as a string or null if the attribute doesn't exist. The syntax is shown below:

```
myNode.getAttribute("AttName");
```

Most HTML attributes are also directly available as properties of the element node; for example, for an `img` tag element, the `width`, `height`, and `source` are available as the `width`, `height`, and `src` properties.

The `element.attributes[]` Property

The `attributes[]` property references the collection of a node's attributes. On the face of it, such a collection could be very useful; however, older versions of Internet Explorer include all possible attributes of a node in its `attributes` collection rather than just those attributes currently used by the node. As such, in practice, it's better to access attributes with the `getAttribute()` method.

Lesson 1, Activity 15: Accessing Nodes by Type, Name or Value

The `element.nodeType` Property

Every node has a `nodeType` property, which contains an integer corresponding to a specific type. For example, 1 is an element node, 2 is an attribute node, 3 is a text node, etc. The W3C DOM specifies a set of constants that correspond to these integers. These constants are properties of every node. Unfortunately, Internet Explorer's DOM doesn't support these constants, so we're stuck with using the integers. For reference, the constants are listed below:

Node Type Constants

Constant	Integer
<code>node.ELEMENT_NODE</code>	1
<code>node.ATTRIBUTE_NODE</code>	2
<code>node.TEXT_NODE</code>	3
<code>node.CDATA_SECTION_NODE</code>	4
<code>node.ENTITY_REFERENCE_NODE</code>	5
<code>node.ENTITY_NODE</code>	6
<code>node.PROCESSING_INSTRUCTION_NODE</code>	7
<code>node.COMMENT_NODE</code>	8
<code>node.DOCUMENT_NODE</code>	9
<code>node.DOCUMENT_TYPE_NODE</code>	10
<code>node.DOCUMENT_FRAGMENT_NODE</code>	11
<code>node.NOTATION_NODE</code>	12

To find all of a node's child *element* nodes, you would loop through its `childNodes` collection checking each node one by one:

```
for (var i=0; i<node.childNodes.length; i++) {
  if (node.childNodes[i].nodeType==1) {
    alert("This is an element node.");
  }
}
```

The `element.nodeName` Property

Every node also has a `nodeName` property. For elements and attributes, the `nodeName` property returns the tag name and attribute name, respectively. For other node types, the `nodeName` property returns a string beginning with a pound sign (#) and indicating the node type (e.g, `#text` or `#comment`).

The `element.nodeValue` Property

The `nodeValue` property is usually used with text nodes and it simply returns the text value of the node. Though, it can also be used to return the value of attributes, comments, processing instructions and CDATA sections. For all other node types, it returns `null`.

Lesson 1, Activity 16: Removing Nodes from the DOM

Element nodes have a `removeChild()` method, which takes a single parameter: the child node to be removed. There is no W3C method for a node to remove itself, but the following function will do the trick:

```
function removeElement(elem) {
    elem.parentNode.removeChild(elem);
    elem=null;
}
```

`removeChild(elem)` only removes the element from its parent. Setting `elem` to `null` destroys the element.

Sometimes it's useful to remove all of a node's children in one fell swoop. The function below will handle this:

```
function removeAllChildren(parent) {
    while (parent.hasChildNodes()) {
        removeElement(parent.childNodes[0]);
    }
}
```

Both of these functions are in our [ClassFiles/lib.js](#) library.

DOM Differences: The Whitespace Problem

The W3C specification is not entirely clear on how certain whitespace should be treated in the DOM. For example, Internet Explorer also ignores whitespace that occurs between open and close tags (e.g., `<td> </td>`). This is just plain wrong, but it doesn't generally create problems.

The whitespace in question is any that occurs between a close tag and an open tag (e.g., `</tr> <tr>`), between two open tags (e.g., `<tr> <td>`), or between two close tags (e.g., `</td> </tr>`). The question is: should this whitespace be preserved or ignored in the DOM. The answer, it turns out, is unclear; however, the way a browser treats this space is very important. To illustrate, open

[HTMLDOM/Demos/Hierarchy.html](#) in Internet Explorer 8 or earlier and click on the middle list item. You will get the following message:

You clicked on a **LI** element.

1. parentNode: UL
2. firstChild: #text
3. lastChild: #text
4. nextSibling: LI
5. previousSibling: LI
6. childNodes.length: 1

Notice that the `nextSibling` and `previousSibling` properties both reference list item nodes, whereas they referenced text nodes in Firefox. If you were to click on the `ul` element itself, you would find that Internet Explorer 8 and earlier consider it to have three child nodes (the list items), whereas Firefox and other modern browsers (including Internet Explorer 9) consider it to have seven. Firefox counts the whitespace-only text nodes. Internet Explorer 8 and earlier do not.

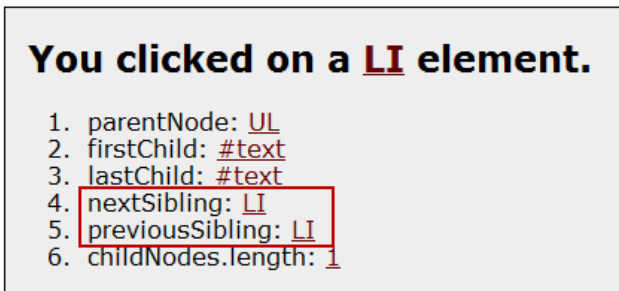
For a lengthy discussion on this topic, see https://bugzilla.mozilla.org/show_bug.cgi?id=26179.

Lesson 1, Activity 19: Equalizing the Browser DOMs

Duration: 15 to 25 minutes.

In this exercise, you will write a function that removes all "empty" child text nodes (text nodes containing only whitespace) from an element. We will use this function to remove discrepancies across browsers caused by the different methods of handling whitespace between elements.

1. Open [HTMLDOM/Exercises/Hierarchy](#) for editing.
2. Notice that when any part of the body is clicked `getNodeInfo()` is called. The `getTarget()` function returns the element that was clicked (i.e, the target element). Don't worry about *how* it works now. We'll show you later in the lesson. Once known, the target element is passed to the `getNodeInfo()` function.
3. Write a function called `removeWhitespace()`. The function should allow for a single node to be passed in and should iterate through that node's `childNodes` collection removing any text node made up of only whitespace (i.e, no non-whitespace characters).
4. Modify the `getNodeInfo()` function so that it makes use of `removeWhitespace()` to clean all sibling and child whitespace-only text nodes of the passed-in node before creating the message.
5. To test your solution, open [HTMLDOM/Exercises/Hierarchy.html](#) in a modern browser (not IE8 or before) and click on the middle list item. Both the previous and next siblings should now be list items as shown below:



Challenge

Make the `removeWhitespace()` function allow for recursive removal of whitespace and use it to remove all the whitespace-only text nodes from the DOM when the page loads.

Solution:

[HTMLDOM/Solutions/Hierarchy.html](#)

```

---- C O D E   O M I T T E D ----

<script type="text/javascript">
observeEvent(window,"load",function() {
  observeEvent(document.body,"click",function(e) {
    target = getTarget(e);
    getNodeInfo(target);
  });
});

function getNodeInfo(target) {
  removeWhitespace(target.parentNode,true);
  removeWhitespace(target,true);
  var msg = "<h2>You clicked on a <span>" + target.nodeName + "</span> element.</h2><ol>";
  msg += "<li>parentNode: <span>" + target.parentNode.nodeName + "</span></li>";
  if(target.firstChild) msg += "<li>firstChild: <span>" + target.firstChild.nodeName + "</span></li>";
  if(target.lastChild) msg += "<li>lastChild: <span>" + target.lastChild.nodeName + "</span></li>";
  if(target.nextSibling) msg += "<li>nextSibling: <span>" + target.nextSibling.nodeName + "</span></li>";
  if(target.previousSibling) msg += "<li>previousSibling: <span>" + target.previousSibling.nodeName + "</span></li>";
  msg += "<li>childNodes.length: <span>" + target.childNodes.length + "</span></li></ol>";
  document.getElementById("output").innerHTML=msg;
}

function removeWhitespace(node,recursive) {
  var childNode;
  for (var i=node.childNodes.length-1; i>=0; i--) {
    childNode = node.childNodes[i];
    if (childNode.nodeType == 3 && !(/\S/.test(childNode.nodeValue))) {
      node.removeChild(childNode);
    }
  }
  if (recursive) {
    for (var i=0; i<node.childNodes.length; i++) {
      removeWhitespace(node.childNodes[i],true);
    }
  }
}

```

```

    }
  }
}
</script>
---- C O D E   O M I T T E D ----

```

Challenge Solution:

HTMLDOM/Solutions/Hierarchy-challenge.html

```

---- C O D E   O M I T T E D ----

observeEvent(window,"load",function() {
  removeWhitespace(document.body,true);
  observeEvent(document.body,"click",function(e) {
    target = getTarget(e);
    getNodeInfo(target);
  });
});
---- C O D E   O M I T T E D ----

function removeWhitespace(node,recursive) {
  var childNode;
  for (var i=node.childNodes.length-1; i>=0; i--) {
    childNode = node.childNodes[i];
    if (childNode.nodeType == 3 && !(/\S/.test(childNode.nodeValue))) {
      node.removeChild(childNode);
    } else if (recursive && childNode.hasChildNodes()) {
      removeWhitespace(childNode,true);
    }
  }
}
---- C O D E   O M I T T E D ----

```

Lesson 1, Activity 21: Creating New Nodes

The document node has separate methods for creating element nodes and creating text nodes: `createElement()` and `createTextNode()`. These methods each create a node in memory that then has to be placed somewhere in the object hierarchy. A new node can be inserted as a child to an existing node with that node's `appendChild()` and `insertBefore()` methods.

You can also use the `appendChild()` and `insertBefore()` methods to move an existing node - the node will be removed from its current location and placed at the new location (since the same node cannot exist twice in the same document).

These methods and some others are described in the table below.

Methods for Inserting Nodes

Method	Description
<code>appendChild()</code>	Takes a single parameter: the node to insert, and inserts that node after the last child node.
<code>insertBefore()</code>	Takes two parameters: the node to insert and the child node that it should precede. The new child node is inserted before the referenced child node.
<code>replaceChild()</code>	Takes two parameters: the new node and the node to be replaced. It replaces the old node with the new node and returns the old node.
<code>setAttribute()</code>	Takes two parameters: the attribute name and value. If the attribute already exists, it replaces it with the new value. If it doesn't exist, it creates it.

The sample below illustrates how these methods work.

Code Sample:

HTMLDOM/Demos/InsertingNodes.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
observeEvent(window,"load",function() {
var appendElem = document.getElementById("append");
var prependElem = document.getElementById("prepend");
var changeElem = document.getElementById("change-num");
var replaceElem = document.getElementById("replace");
observeEvent(appendElem,"click",appendNewListItem,false);
observeEvent(prependElem,"click",prependNewListItem,false);
observeEvent(changeElem,"click",changeStartNum,false);
observeEvent(replaceElem,"click",replaceOlWithUl,false);
});

function appendNewListItem() {
var li = document.createElement("li");
var liText = document.createTextNode("New List Item");
li.appendChild(liText);
document.getElementById("first-list").appendChild(li);
}

function prependNewListItem() {
var li = document.createElement("li");
var liText = document.createTextNode("New List Item");
var firstList = document.getElementById("first-list");
li.appendChild(liText);
firstList.insertBefore(li,firstList.firstChild);
}

function changeStartNum() {
var firstList = document.getElementById("first-list");
```

```

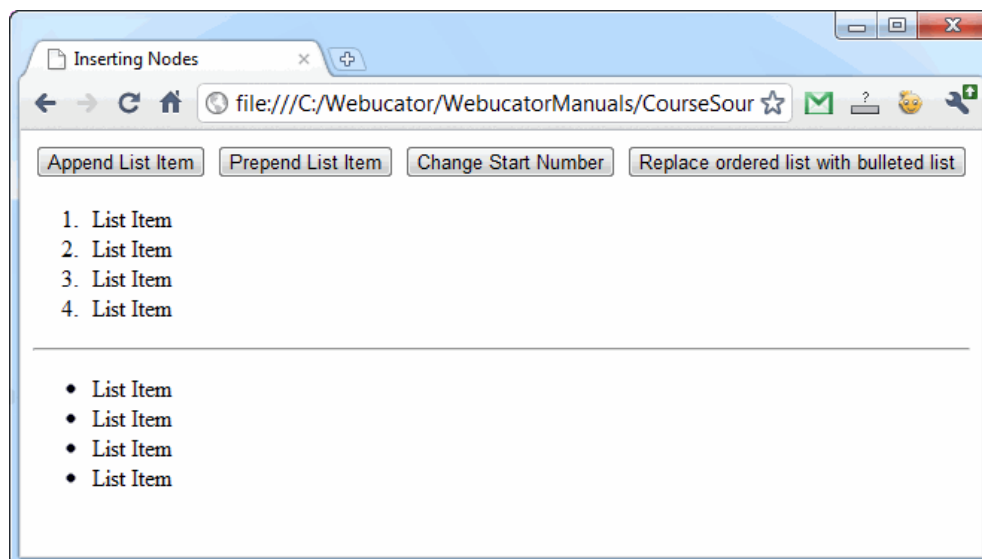
var start=firstList.getAttribute("start");
if (start) {
    start++;
} else {
    start=2;
}
firstList.setAttribute("start",start);
}

function replaceOlWithUl() {
    list = document.getElementById("first-list");
    list2 = document.getElementById("second-list");
    list.parentNode.replaceChild(list2,list);
    list = list2;
}
</script>
<title>Inserting Nodes</title>
</head>

<body>
<form id="menu">
    <button id="append" type="button">Append List Item</button>
    <button id="prepend" type="button">Prepend List Item</button>
    <button id="change-num" type="button">Change Start Number</button>
    <button id="replace" type="button">Replace ordered list with bulleted list</button>
</form>
<ol id="first-list">
    <li>List Item</li>
    <li>List Item</li>
    <li>List Item</li>
    <li>List Item</li>
</ol>
<hr>
<ul id="second-list">
    <li>List Item</li>
    <li>List Item</li>
    <li>List Item</li>
    <li>List Item</li>
</ul>
</body>
</html>

```

The page is shown below in a browser. Click on any of the menu items to see the methods in action. The unordered list will move when it replaces the ordered list, so it will then appear above the horizontal rule line.



As a coding practice, it is better to create a structure before inserting it into the page than to put the parent element in the page and then append to

it. This saves the browser from unnecessary redrawing of the page as each child is added.

A Note on the `setAttribute()` Method

You can use the `setAttribute()` method to change the value of all attributes by name as you would expect with one exception. In Internet Explorer 8 and earlier, the `class` attribute must be referred to as `"className"` in the `setAttribute()` method. This means that you shouldn't use `setAttribute()` for setting the class. Instead, change the `className` property.

```
node.className = "new-class-name";
```

Or you can use the `addClass()` function in our [ClassFiles/lib.js](#) file.

Lesson 1, Activity 22: Identifying the Target of an Event

Often you will want to take action on the target of an event. For example, in a drag-and-drop application, you click down on an element and drag it around the screen. To write that code, you need to be able to identify which element receives the click.

In most modern browsers, this is done with the `target` property of the event itself. The following demo illustrates:

Code Sample:

HTMLDOM/Demos/target.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" type="text/css" href="../Styles/propagation.css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
  observeEvent(window,"load",function() {
    var outer=document.getElementById("outer");
    var inner=document.getElementById("inner");
    var heading=document.getElementById("heading");
    observeEvent(outer,"click",function(e) {
      findTarget(e);
    }, false);
    observeEvent(inner,"click",function(e) {
      findTarget(e);
    }, false);
    observeEvent(heading,"click",function(e) {
      findTarget(e);
    }, false);
  });

  function findTarget(e) {
    e.target.innerHTML += " | click | ";
    stopPropagation(e);
  }
</script>
<title>Target</title>
</head>
<body>
  <h1 id="heading">Target</h1>
  <div id="outer">outer
    <div id="inner">inner</div>
  </div>
</body>
</html>
```

In your browser, click on any of the elements: **outer**, **inner**, or the h1 element and the word "click" will get added to the element.

Unfortunately, if you try this in IE8 and earlier, you will get an error something like *"'target' is null or not an object"*. IE8 and earlier use a different property to identify the target element: `srcElement`.

In ClassFiles/lib.js, there is a cross-browser `getTarget()` function shown below.

```
function getTarget(e) {
  e = e || window.event;
  var target = e.target || e.srcElement;
  return target;
}
```

Lesson 1, Activity 23: Creating and Inserting DOM Nodes

Duration: 45 to 60 minutes.

In this exercise, you will modify our soccer game so that it adds new balls when the user misses.

1. Open [HTMLDOM/Solutions/soccer-game-multiball.html](#) in your **browser** and play for a bit. Notice that when you miss the ball, a new one appears. Also notice that when you hit a ball, it gets removed from the field and added to the cage using `appendChild()`.
2. Open [HTMLDOM/Exercises/soccer-game-multiball.html](#) in your **editor**. Notice we have changed the HTML some:
 1. Added a "msg" div to hold messages (no more alerts).
 2. Added a "cage" div to hold balls that have been removed from the field.
 3. Added a "level" span to show what level the user is on. This is for the challenge.
 4. Removed the "click-count" span. We will show the click-count messages in the "msg" div.
 5. Removed the img. We'll add it using JavaScript when the game starts.
3. We have also changed the JavaScript some:
 1. Added a `msg()` function to display messages in our new "msg" div.
 2. Modified the `moveBall()` function to put a red border around the ball that escapes the field and to use the `msg()` function to report the end of the game.
 3. Added a `removeBall()` function, which we'll discuss later.
4. We're going to have multiple balls and associated timers now, so the first thing you need to do is change the scalar global variable `timer` to an empty `timers` array and add an empty `balls` array.
5. Also add a global variable `clickCount` and set it to 0.
6. When the user missed the ball, we used to just alert "Miss!". Now we want to do much more. Write an `addBall()` function that does the following:
 1. Create a "ball" `img` element and append it to the global `balls` array.
 2. Set a `ballNum` variable to hold the length of the `balls` array.
 3. Set the following properties of the new `img` element:
 - `src: "../Images/ball.gif"`
 - `alt: "Ball " + ballNum`
 - `title: "Ball " + ballNum`
 - `className: "ball"`
 - `style.top: "92px"`
 - `style.left: "92px"`
 4. Append the new "ball" `img` to the "field" div so the ball appears on the field.
 5. Use our `observeEvent()` function to attach the `hit()` `mousedown` events on the new ball.
 6. Append an `Interval` object to the `timers` array to set the ball moving: `timers.push(setInterval(function() { moveBall(ball) },20));`
7. In the `miss()` function, remove the `alert()` and replace it with a call to `addBall()`.
8. Replace the existing `incrementClickCount()` function with this one:

```
function incrementClickCount() {
  clickCount++;
  msg(clickCount + " clicks");
}
```

9. Replace the existing `start()` function with this one:

```
function start() {
  clickCount=0;
  removeBalls();
  addBall();
  document.getElementById("start").disabled=true;
  observeEvent(document.getElementById("field"), "click", miss, false);
}
```

1. Notice we call the `addBall()` function we just wrote.
2. We call `removeBalls()` before adding a ball. That's because there might be left over balls from the previous game. You'll need to write that function.
3. Remember that we put code in the `addBall()` function to observe events on each new ball we add. Therefore, we don't need to observe events on the ball in the `start()` function.

10. Write a `removeBalls()` function that removes all balls from the "field" div and empties the global `balls` array.
11. We are all set for starting the game and handling misses. Now we need to add code to capture hits. In our simple game, there was only one ball. When the user hit it the game ended. But now, the game is more complicated. There can be many balls on the field at once. When the user clicks on one it must be moved into the cage. The game is won only when the last ball on the field is hit.
12. The first thing you need to do in the `hit()` function is figure out which ball was hit. In other words, which ball was the **target** of the mousedown event. **Hint:** remember the `getTarget()` function in [ClassFiles/lib.js](#).
13. You then need to move that ball to the cage. Call the `removeBall()` function, which is already included in your code, to do so.
 1. The actual removal of the ball to the cage is pretty straightforward:

```
var cage=document.getElementById("cage");
cage.appendChild(ball);
```

An element can only be in one place in the DOM, so when we we append it to the "cage" div, it gets removed from the "field" div.

2. The stopping of the timer is a bit messy. We need to stop the timer that is associated with moving the ball that gets removed, but there is no direct relationship between the ball and the timer. We have stored both balls and timers in arrays, such that the *n*th timer calls the code that moves the *n*th ball. That kind of code is precarious at best. We'll learn how to make it better when we get to object-oriented JavaScript. For the time being, we need to find out which ball in the `balls` array is being removed so we can clear the timer at the same index in the `timers` array. We use the `alt` value to do this:

```
var timerNum = ball.alt.split(' ')[1]-1;
clearInterval(timers[timerNum]);
```

We have to subtract 1 because JavaScript arrays are base 0. Note that if we didn't clear the timer, the ball would continue to move around in the cage.

3. Don't worry about the `style` properties. We'll address that sort of thing when we get to Dynamic HTML.
14. Next, back in the `hit()` function, increment the click count and stop the event from propagating to the field.
15. The last thing to do in the `hit()` function is to determine if the "field" div has child nodes.
 - If it does not have child nodes, output a message (using `msg()`) that reads "Hit! Congrats! You got it in " + `clickCount` + " clicks." And call `endGame()`.
 - If it does have child nodes, output a message (again using `msg()`) that reads "Congrats! You got one."
16. Finally, you must modify the `endGame()` function as follows:
 1. Clear all the timers and empty the `timers` array.
 2. Stop observing events for all the balls remaining on the field and empty the `balls` array.

Code Sample:

[HTMLDOM/Exercises/soccer-game-multiball.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" type="text/css" href="../Styles/soccer-game.css">
<script type="text/javascript" src="../../lib.js"></script>
<script type="text/javascript">
var timer = null;
observeEvent(window,"load",function() {
  observeEvent(document.getElementById("start"),"click",start,false);
});

function miss() {
  incrementClickCount();
  alert("Miss!");
}

function hit(e) {
  incrementClickCount();
  alert("Hit! Congrats! You got it in " + document.getElementById("click-count").innerHTML + " clicks.");
  stopPropagation(e);
  clearInterval(timer);
  endGame();
}

function removeBall(ball) {
```

```

var timerNum = ball.alt.split(' ')[1]-1;
var cage=document.getElementById("cage");
clearInterval(timers[timerNum]);
cage.appendChild(ball);
ball.style.position="relative";
ball.style.left="0px";
ball.style.top="0px";
}

function start() {
var ball = document.getElementById("ball");
document.getElementById("start").disabled=true;
ball.style.top="92px";
ball.style.left="92px";
document.getElementById("click-count").innerHTML="0";
timer=setInterval(function() { moveBall(ball) },20);

observeEvent(document.getElementById("field"),"click",miss,false);
observeEvent(document.getElementById("ball"),"mousedown",hit,false);
}

function msg(msg) {
document.getElementById("msg").innerHTML=msg;
}

function moveBall(ball) {
var x=2,y=2;
var left, top;
if (Math.floor(Math.random()*2)==0) {
x = -x;
}
if (Math.floor(Math.random()*2)==0) {
y = -y;
}
left = parseInt(ball.style.left);
top = parseInt(ball.style.top);
if (top < 0 || top > 184 || left < 0 || left > 184) {
ball.style.border = "1px solid red";
msg("Game over!  
>" + ball.alt + " is at pos x:" + left + ", y:" + top);
endGame();
}
ball.style.left = (left + x) + "px";
ball.style.top = (top + y) + "px";
}

function incrementClickCount() {
var clickCount = document.getElementById("click-count").innerHTML;
clickCount++;
document.getElementById("click-count").innerHTML = clickCount;
}

function endGame() {
unobserveEvent(document.getElementById("field"),"click",miss,false);
unobserveEvent(document.getElementById("ball"),"mousedown",hit,false);
document.getElementById("start").disabled=false;
}
</script>
<title>Click the Ball</title>
</head>
<body>
<div id="container">
<h1>Click the Ball</h1>
<div>
<button id="start">Start</button>
<span id="level">Level 1</span>
</div>
<div id="msg">Click the ball to make it disappear. If you miss, another ball will be added.</div>
<div id="field"></div>
<div id="cage">
<h3>CAGE</h3>
</div>
</div>

```

```
</body>
</html>
```

Challenge

Just in case this exercise isn't challenging *enough*, try adding levels so that when the user succeeds at removing the balls from the field, he or she moves to the next level, in which two balls are added at the start and with each miss. Then three balls, then four, etc. Open HTMLDOM/Solutions/soccer-game-multiball-challenge.html in your browser to see how the game should work.

Solution:

HTMLDOM/Solutions/soccer-game-multiball.html

```
---- C O D E   O M I T T E D ----

<script type="text/javascript">
var timers = [];
var balls = [];
var clickCount=0;
observeEvent(window,"load",function() {
  observeEvent(document.getElementById("start"),"click",start,false);
});

function miss() {
  incrementClickCount();
  addBall();
}

function hit(e) {
  e = e || window.event;
  var ball = getTarget(e);
  var field = ball.parentNode;
  removeBall(ball);
  incrementClickCount();
  stopPropagation(e);
  if (!field.hasChildNodes()) {
    msg("Hit! Congrats! You got it in " + clickCount + " clicks.");
    endGame();
  } else {
    msg("Congrats! You got one.");
  }
}

function removeBall(ball) {
  var timerNum = ball.alt.split(' ')[1]-1;
  var cage=document.getElementById("cage");
  clearInterval(timers[timerNum]);
  cage.appendChild(ball);
  unobserveEvent(ball,"mousedown",hit,false);
  ball.style.position="relative";
  ball.style.left="0px";
  ball.style.top="0px";
}

function start() {
  clickCount=0;
  removeBalls();
  addBall();
  document.getElementById("start").disabled=true;
  observeEvent(document.getElementById("field"),"click",miss,false);
}

function addBall() {
  var field=document.getElementById("field");
  var ball=document.createElement("img");
  balls.push(ball);
  var ballNum = balls.length;
  ball.src="../Images/ball.gif";
```

```

ball.alt="Ball " + ballNum;
ball.title="Ball " + ballNum;
ball.className="ball";
ball.style.top="92px";
ball.style.left="92px";
field.appendChild(ball);
observeEvent(ball,"mousedown",hit,false);
timers.push( setInterval(function() { moveBall(ball) },20) );
}

function msg(msg) {
  document.getElementById("msg").innerHTML=msg;
}

function moveBall(ball) {
  var x=2,y=2;
  var left, top;
  if (Math.floor(Math.random()*2)==0) {
    x = -x;
  }
  if (Math.floor(Math.random()*2)==0) {
    y = -y;
  }
  left = parseInt(ball.style.left);
  top = parseInt(ball.style.top);
  if (top < 0 || top > 184 || left < 0 || left > 184) {
    ball.style.border = "1px solid red";
    msg("Game over!<br>" + ball.alt + " is at pos x:" + left + ", y:" + top);
    endGame();
  }
  ball.style.left = (left + x) + "px";
  ball.style.top = (top + y) + "px";
}

function removeBalls() {
  var field=document.getElementById("field");
  removeAllChildren(field);
  balls=[];
}

function incrementClickCount() {
  clickCount++;
  msg(clickCount + " clicks");
}

function endGame() {
  var field=document.getElementById("field");
  var remainingBalls=field.childNodes;
  unobserveEvent(document.getElementById("field"),"click",miss,false);
  for (var t=0; t<timers.length; ++t) {
    clearInterval(timers[t]);
  }
  timers=[];
  for (var b=0; b<remainingBalls.length; b++) {
    unobserveEvent(remainingBalls[b],"mousedown",hit,false);
  }
  balls=[];
  document.getElementById("start").disabled=false;
}
</script>
<title>Click the Ball</title>
</head>
<body>
<div id="container">
  <h1>Click the Ball</h1>
  <div>
    <button id="start">Start</button>
    <span id="level">Level 1</span>
  </div>
  <div id="msg">Click the ball to make it disappear. If you miss, another ball will be added.</div>
  <div id="field"></div>
  <div id="cage">

```

```

    <h3>CAGE</h3>
  </div>
</div>
</body>
</html>

```

Solution:

HTMLDOM/Solutions/soccer-game-multiball-challenge.html

```

---- C O D E   O M I T T E D ----

<script type="text/javascript">
var timers = [];
var balls = [];
var clickCount=0;
var level=1;
observeEvent(window,"load",function() {
  observeEvent(document.getElementById("start"),"click",start,false);
});

function miss() {
  incrementClickCount();
  for (var i=0; i<level; i++) {
    addBall();
  }
}

function hit(e) {
  e = e || window.event;
  var ball = getTarget(e);
  var field = ball.parentNode;
  removeBall(ball);
  incrementClickCount();
  stopPropagation(e);
  if (!field.hasChildNodes()) {
    msg("Hit! Congrats! You got it in " + clickCount + " clicks.");
    incrementLevel();
    endGame();
  } else {
    msg("Congrats! You got one.");
  }
}

function incrementLevel() {
  level++;
  document.getElementById("level").innerHTML="Level " + level;
}

function removeBall(ball) {
  var timerNum = ball.alt.split(' ')[1]-1;
  var cage=document.getElementById("cage");
  clearInterval(timers[timerNum]);
  cage.appendChild(ball);
  unobserveEvent(ball,"mousedown",hit,false);
  ball.style.position="relative";
  ball.style.left="0px";
  ball.style.top="0px";
}

function start() {
  document.getElementById("level").innerHTML="Level " + level;
  clickCount=0;
  removeBalls();
  for (var i=0; i<level; i++) {
    addBall();
  }
  document.getElementById("start").disabled=true;
  observeEvent(document.getElementById("field"),"click",miss,false);
}

```

```

function addBall() {
  var field=document.getElementById("field");
  var ball=document.createElement("img");
  balls.push(ball);
  var ballNum = balls.length;
  ball.src="../Images/ball.gif";
  ball.alt="Ball " + ballNum;
  ball.title="Ball " + ballNum;
  ball.className="ball";
  ball.style.top="92px";
  ball.style.left="92px";
  field.appendChild(ball);
  observeEvent(ball,"mousedown",hit,false);
  timers.push( setInterval(function() { moveBall(ball) },20) );
}

function msg(msg) {
  document.getElementById("msg").innerHTML=msg;
}

function moveBall(ball) {
  if(ball.parentNode.id=="cage") {
    ball.style.border="1px green solid";
    return;
  }
  var x=2,y=2;
  var left, top;
  if (Math.floor(Math.random()*2)==0) {
    x = -x;
  }
  if (Math.floor(Math.random()*2)==0) {
    y = -y;
  }
  left = parseInt(ball.style.left);
  top = parseInt(ball.style.top);
  if (top < 0 || top > 184 || left < 0 || left > 184) {
    ball.style.border = "1px solid red";
    msg("Game over!<br>" + ball.alt + " is at pos x:" + left + ", y:" + + top);
    level=1;
    endGame();
  }
  ball.style.left = (left + x) + "px";
  ball.style.top = (top + y) + "px";
}

}

---- C O D E   O M I T T E D ----

<div>
  <button id="start">Start</button>
  <span id="level">Level 1</span>
</div>
---- C O D E   O M I T T E D ----

```