Internet Computing

Dynamic Documents with JavaScript

Dynamic HTML

- An HTML document whose content, structure, and look could be changed after the document has been and is still being displayed by a browser
  - Use JavaScript to create dynmaic HTML

JavaScript Execution Environment (inside the browser)

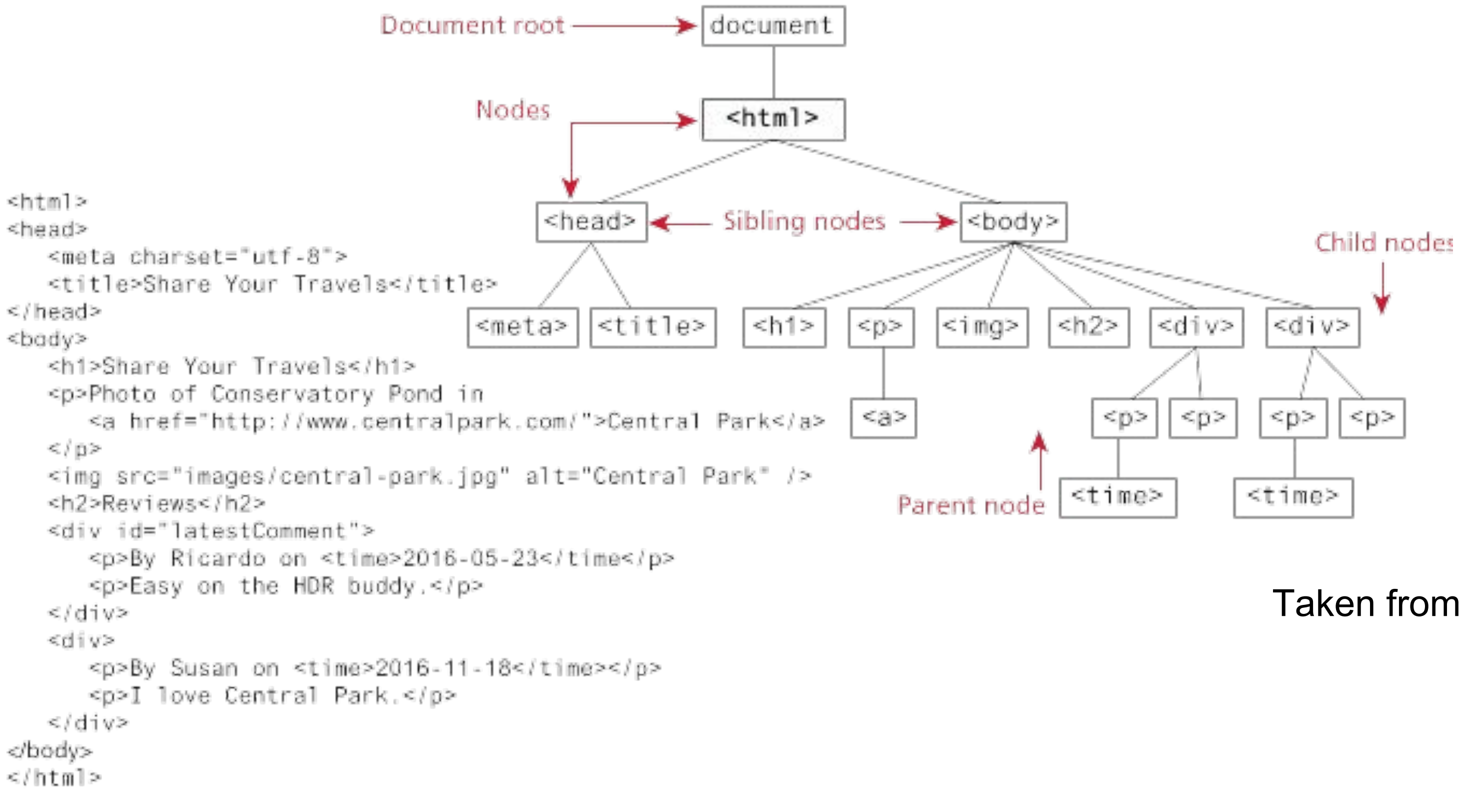
- The JavaScript `Window` object represents the window in which the browser displays documents
  - The browser is modeled as an object and the document it displays is modeled an object
    - The navigation button, the URL, ..
  - The Window object provides the largest enclosing referencing environment for scripts
    - All global variables are properties of Window
  - document
    - On of the properties of Window onect:
    - a reference to the Document object that the window displays
    - Contains the current html document: window.document
      - Since JavaScript lives inside the window, we can get the document by calling it directly

JavaScript Execution Environment (inside the browser)

- The Document Object Model (DOM)
  - The model of the document that forms the current web page
  - Som properties of 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
    - Other properties: `anchors`, `links`, and `images`

  - It is an abstract model that defines the interface between HTML documents and application programs—an API
  - HTML elements are represented as objects and element attributes are represented as properties

The Document Object Model (DOM)



```
<html>
<head>
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>2016-05-23</time</p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

Taken from [2]

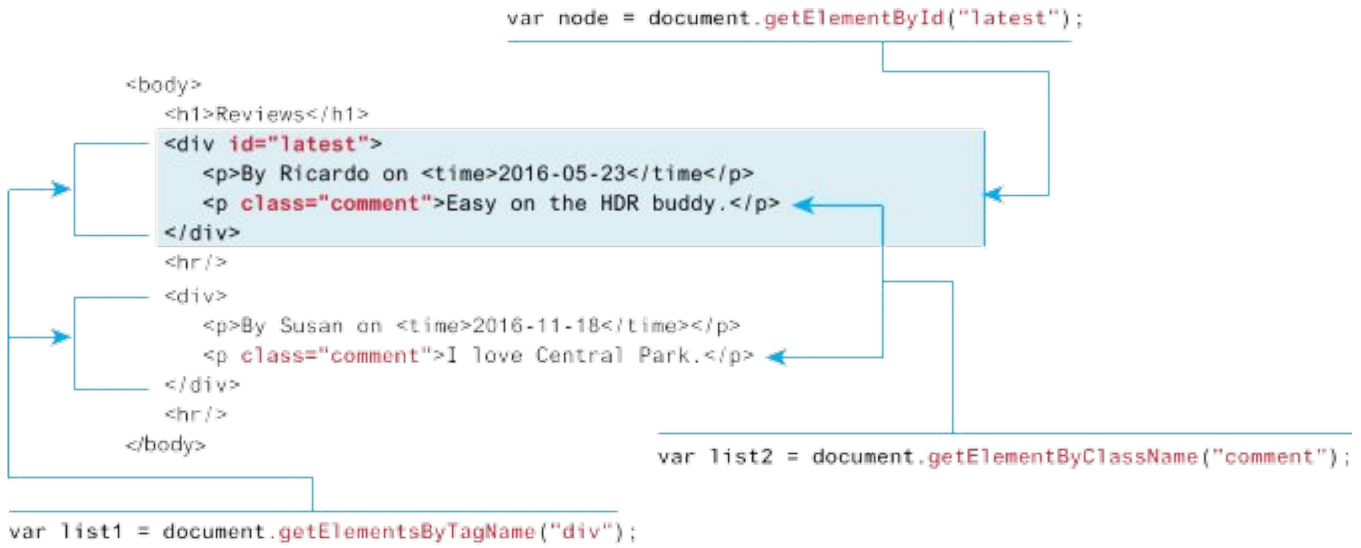
DOM Standards

- DOM 0 is supported by all JavaScript-enabled browsers (no written specification)
- DOM 1 was released in 1998
- DOM 2 was released in 2000
  - Defined how style information attached to a document can be manipulated
  - Provided document traversal and a complete event model
- DOM 3 is the latest approved standard (2004)

Access Elements in DOM

- Access Properties of document
    - Document.body
    - Document.title
    - Document.URL
- Access a node or group of nodes, use methods available to document
    - Specific methods
        - getElementById()
        - getElementsByTagName()
        - getElementsByClassName()
    - Newer methods: access documents similar to CSS selectors
        - querySelector()
        - querySelectorAll()
        - You can pass a CSS selectors to these methods

Access Elements in DOM



```
var node = document.getElementById("latest");

<body>
   <h1>Reviews</h1>
   <div id="latest">
      <p>By Ricardo on <time>2016-05-23</time</p>
      <p class="comment">Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p class="comment">I love Central Park.</p>
   </div>
   <hr/>
</body>

var list2 = document.getElementByClassName("comment");

var list1 = document.getElementsByTagName("div");
```

Taken from [2]

Access Elements in DOM

- DOM address
  - Position of elements in the document
  - Problem: document changes
- Element name
  - requires the element and all of its ancestors (except body) to have name attributes
  - XHTML 1.1 does not allow name attribute in form

```
<form action = "">
      <input type = "button" name = "pushMe" />
</form>
Document.forms[0].elements[0]
```

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

Access Elements in DOM

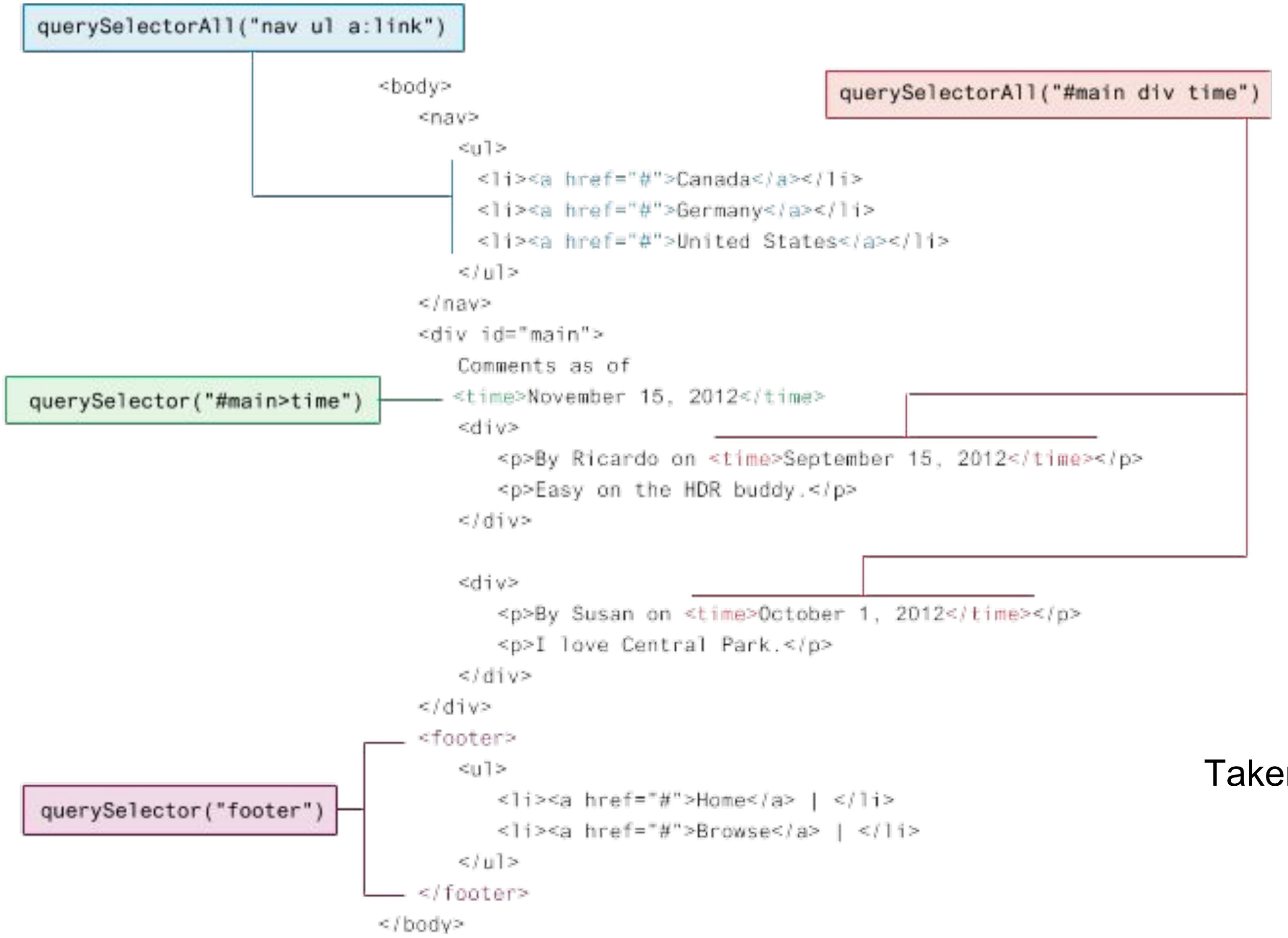
- getElementById Method
  - defined in DOM 1

```
<form action = "">
       <input type = "button"  id = "pushMe" /
 </form>
document.getElementById("pushMe")
```

- Checkboxes and radio button have an implicit array, which has their name

```
<form id = "topGroup">
  <input type = "checkbox"  name = "toppings"
        value = "olives" />
  ...
  <input type = "checkbox"  name = "toppings"
        value = "tomatoes" />
</form>

var numChecked = 0;
var dom = document.getElementById("topGroup");
for index = 0; index < dom.toppings.length;  index++)
          if (dom.toppings[index].checked]
                    numChecked++;
```


querySelectorAll("nav ul a:link")
querySelectorAll("#main div time")
querySelector("#main>time")
querySelector("footer")

```
<body>
   <nav>
      <ul>
         <li><a href="#">Canada</a></li>
         <li><a href="#">Germany</a></li>
         <li><a href="#">United States</a></li>
      </ul>
   </nav>
   <div id="main">
      Comments as of
      <time>November 15, 2012</time>
      <div>
         <p>By Ricardo on <time>September 15, 2012</time></p>
         <p>Easy on the HDR buddy.</p>
      </div>

      <div>
         <p>By Susan on <time>October 1, 2012</time></p>
         <p>I love Central Park.</p>
      </div>
   </div>
   <footer>
      <ul>
         <li><a href="#">Home</a> | </li>
         <li><a href="#">Browse</a> | </li>
      </ul>
   </footer>
</body>
```

Taken from [2]

The Document Object Model: Element Node

- Element Node object represents an HTML element contained between the opening <> and closing </> tags for this element. Every node has properties:
    - classList
    - className
    - Id
    - innerHTML
    - Style
    - tagName
    - Full list of properties and methods available at
        - https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement
-

DOM: Access and Change Elements

- To access and changed the properties of an Element, within a document
    - access the Element, with a query selector
    - change the property.
    - Or change the method
        - Not all properties are writeable

DOM: Access and Change Attributes

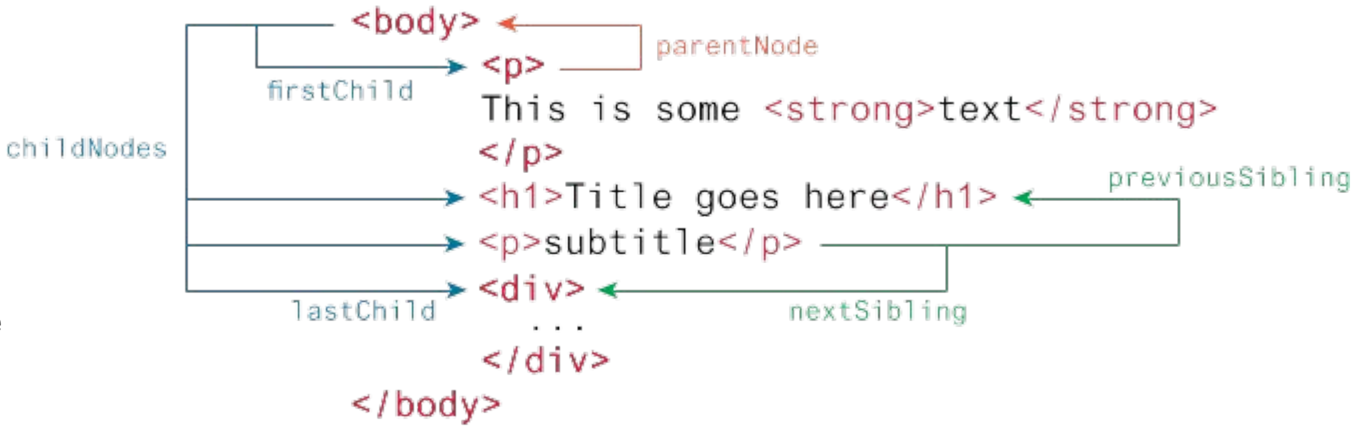
`Element.getAttribute()`

`Element.setAttribute()`

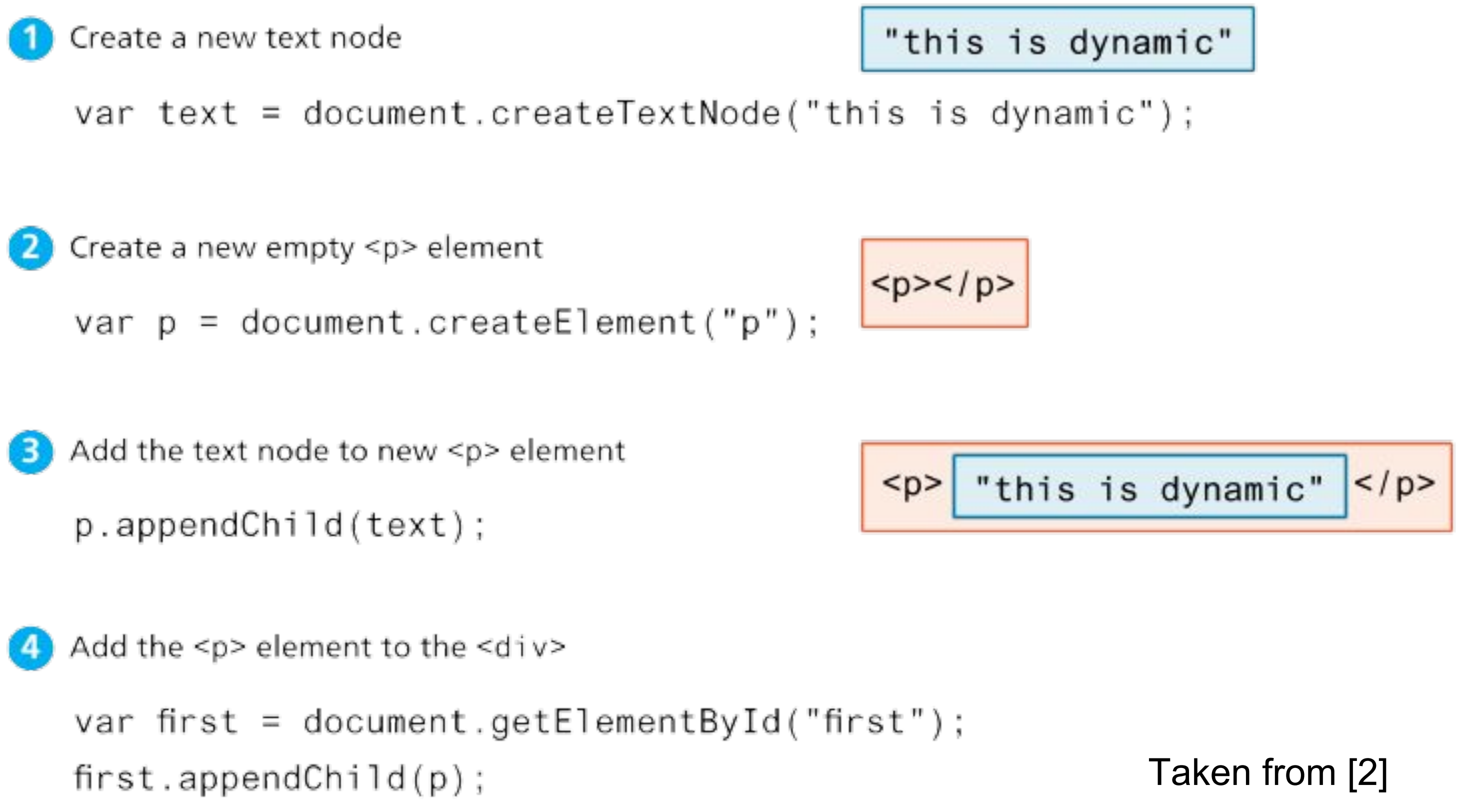
DOM: Modify the DOM

- innerHTML and outerHTML properties

- DOM Tree Traversal
  - Traverse and modify DOM tree structure



Modifying the DOM: Creating and Adding Nodes



1 Create a new text node

```
var text = document.createTextNode("this is dynamic");
```

2 Create a new empty <p> element

```
var p = document.createElement("p");
```

3 Add the text node to new <p> element

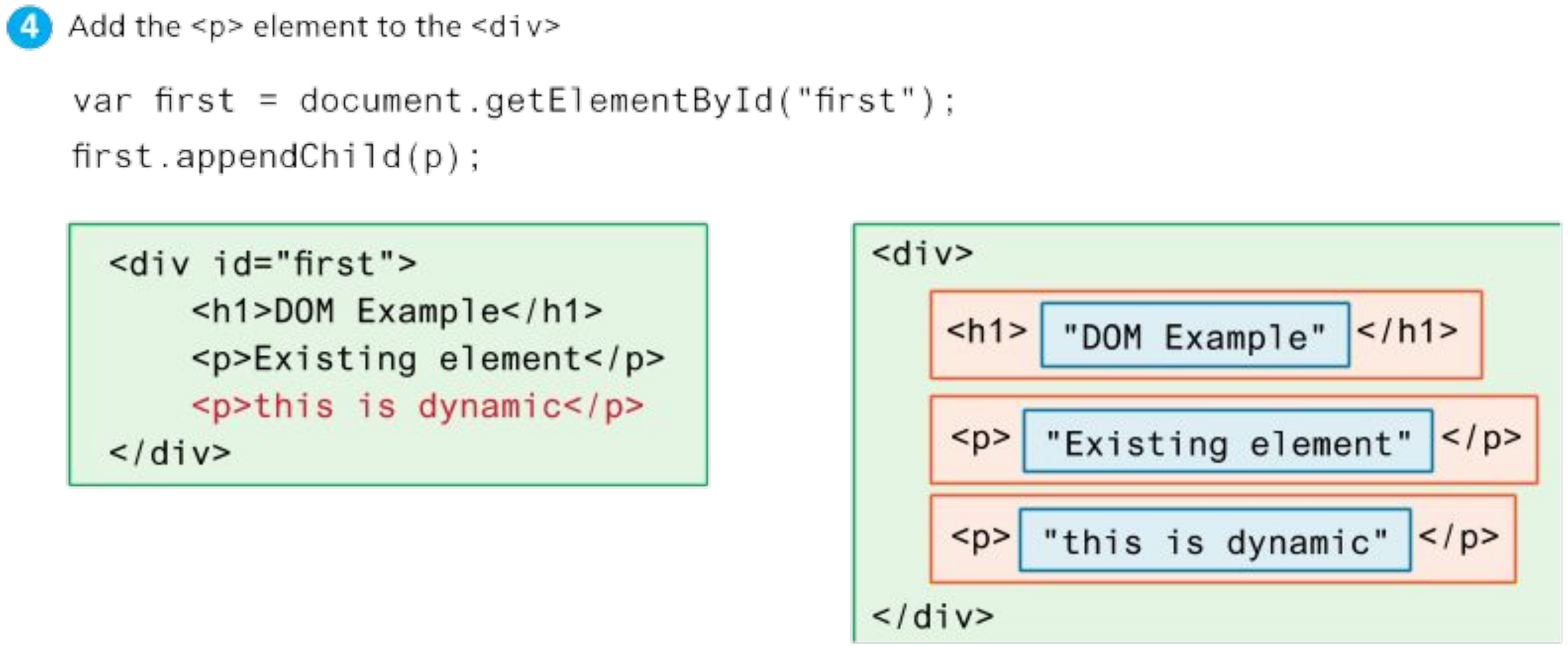
```
p.appendChild(text);
```

4 Add the <p> element to the <div>

```
var first = document.getElementById("first");
first.appendChild(p);
```

Taken from [2]

Modifying the DOM: Creating and Adding Nodes



4 Add the <p> element to the <div>

```
var first = document.getElementById("first");
first.appendChild(p);
```

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
    <p>this is dynamic</p>
</div>
```

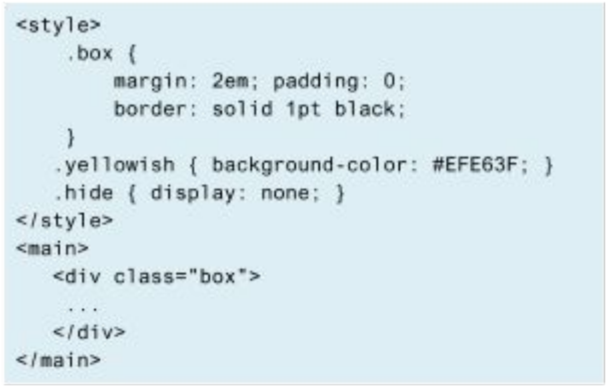
Taken from [2]

DOM: Changing an Element’s Style

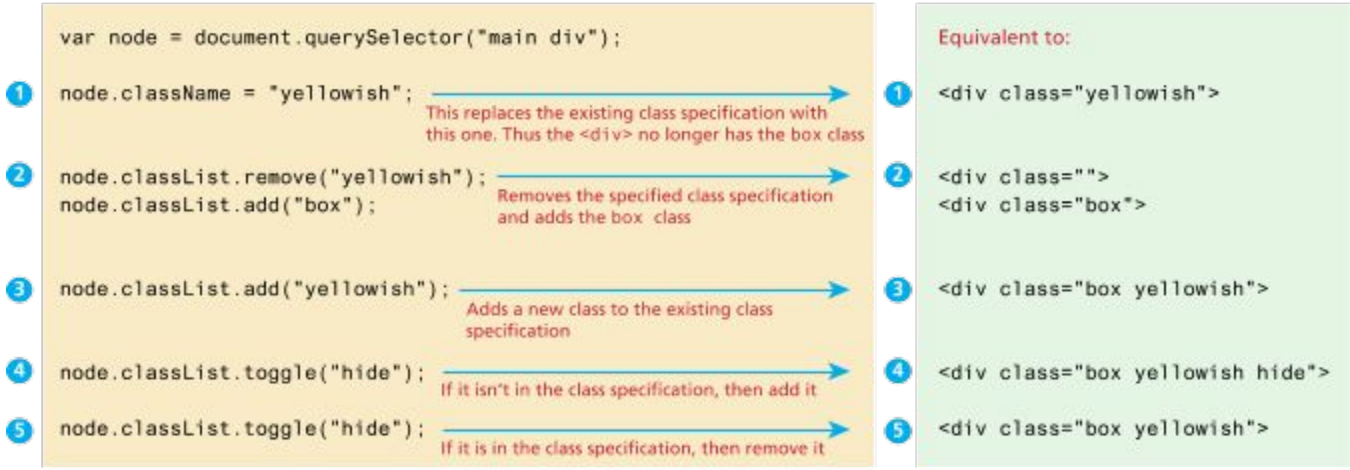
- Only inline CSS is applied
  - When we apply styles to an element using JavaScript the browser adds inline CSS to that element by accessing the style attribute
    - JavaScript does not write to standalone style sheets or any other CSS on the page.
    - These inline styles are stored in the style property, which can be accessed like any other element property.
  - CSSStyleDeclaration: lists every possible property that could be applied using inline CSS
- https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style

DOM:

- Changing an element’s style
  - classList methods



```
<style>
    .box {
        margin: 2em; padding: 0;
        border: solid 1pt black;
    }
   .yellowish { background-color: #EFE63F; }
   .hide { display: none; }
</style>
<main>
   <div class="box">
    ...
   </div>
</main>
```



```
var node = document.querySelector("main div");

1 node.className = "yellowish";
2 node.classList.remove("yellowish");
  node.classList.add("box");
3 node.classList.add("yellowish");
4 node.classList.toggle("hide");
5 node.classList.toggle("hide");
```

This replaces the existing class specification with this one. Thus the <div> no longer has the box class
Removes the specified class specification and adds the box class
Adds a new class to the existing class specification
If it isn't in the class specification, then add it
If it is in the class specification, then remove it

Equivalent to:

```
1 <div class="yellowish">
2 <div class="">
  <div class="box">
3 <div class="box yellowish">
4 <div class="box yellowish hide">
5 <div class="box yellowish">
```

Events

- Browser-level events
  - Load - when the resources finished loading
  - Error - When the resource failed to load
  - Resize - viewport is resized
  - Scroll - viewport is scrolled up/down/left/right
- DOM events
  - Focus - When an element receives focus
  - Blur - when an element loses focus (e.g, form)
  - Reset/submit - form-specific events
  - Mouse events - click, mouseover, drag, drop
- Full list available: https://developer.mozilla.org/en-US/docs/Web/Events

Events

- A notification that something specific has occurred, either with the browser or an action of the browser user
  - An event is an object implicitly created by the browser and the JavaScript system in response to something having happened
- An event handler is a script that is implicitly executed in response to the appearance of an event
- 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 display

Events

- When an event occurs, an event object is created by the browser which has some information about the event
    - MouseEvent e:
        - e.clientX and e.clientY x and y coordinates of the mouse cursor relative to the upper left corner of the client area of the browser window
        - e.target a reference to the target node
            - A reference to the object that generated the event
- https://developer.mozilla.org/en-US/docs/Web/Events

Events: DOM 0 model

- Most commonly used events and their associated tags
- The same attribute can appear in several different tags
  - e.g., The onclick attribute can be in <a> and <input>

Event	**Tag Attribute**
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
Mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

Events handling: DOM 0 model

- Event handlers can be registered in two ways:
  - Inline hooks:
    - Handler is assigned as a string literal to the event's associated attribute in the element
  - Event Property approach
    - The name of event handler function can be assigned to the event property of the JavaScript object associated with the HTML element.

```
onclick = "alert('Mouse click!');"
onclick = "myHandler();"
onload="load_greeting();"
```

```
var dom = document.getElementById("myForm")
dom.elements[0].onclick = planeChoice;
```

Events: examples

- Handling events from body element
  - Example: the load event - triggered when the loading of a document is completed
  - DEMO: load.html, load.js

Events: examples

- Handling events from button elements
  - use the onclick property- Radio buttons
    - DEMO: radio_click.html, radio_click.js
      - the particular button that was clicked can be sent to the handler as a parameter
  - Assign the handler function to the event property of the JavaScript object associated with the HTML element.
    - DEMO radio_click2.html, radio_click2r.js
      - for a radio button group, each element of the array must be assigned
      - the checked property of a radio button object is used to determine whether a button is clicked
      - The code to register handler must follow both the handler function and the HTML form specification so that JavaScript sees both before assigning the property.

Events

- Event handlers can be registered in two ways (DOM 0):
  - By assigning the event handler script to an event tag attribute
  - Assign the address of the handler function to the event property of the JavaScript object associated with the HTML element.
    - Disadvantages:
      - there is no way to use parameters
    - Advantages:
      - It is good to keep HTML and JavaScript separate
      - The handler could be changed during use

Events

- Handling Events from Textbox
  - The focus event
  - SHOW nochange.html & nochange.js

Events

- Validating form input
    - A good use of JavaScript, because it finds errors in form input before it is sent to the server for Processing:
        - Saves server time
        - Save network time
        - Results in quicker response to the user
- Things that must be done:
        - Detect the error and produce an alert message
        - inform the user of the error and present the correct format

Events: Examples

- Validating form input
  - DEMO pswd_chkr.js
    - The form just has two password input boxes to get the passwords and Reset and Submit buttons
    - The event handler is triggered by the submit button and the blur event
  - To keep the form active after the event handler is finished, the handler must return false
    - If an event handler returns false, that tells the browser not to perform any default actions of the events
      - e.g a click on the submit button, the default action is to submit the form data to the server for processing. If user input is being validated in an event handler that is called when the submit event occurs and some of the input is incorrect, the handler return false to avoid sending bad data to the server

Events: Examples

- Validating form input
  - DEMO validator.html
  - The change event: happens when the value of the text boxes are changed and lose focus
  - 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

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

The DOM 2 Event Model

- Event handler registration is done with the addEventListener method
  - Three parameters:
    - Name of the event, as a string literal
    - The handler function
    - A Boolean value that specifies whether the event is enabled during the capturing phase
      - If true, it can only be called during capturing
  - The `currentTarget` property of event always references the object on which the handler is being executed
      - `this`: A reference to the target node in DOM 0 model
  - A temporary handler can be created by registering it and then unregistering it with removeEventListener

The DOM 2 Event Model

- Event propagation:
  - There are three phases for handling events:
    - Capturing phase
      - Events begin at the root and move toward the target node
      - Registered and enabled event handlers at nodes along the way are run
    - Target mode phase
      - If there are registered and not enabled handlers there for the event, they are run
    - Bubbling phase
      - Event goes back to the root; all encountered registered but not enabled handlers are run

DOM 2 Event Model: Example

- DEMO validator2.html
- A revision of validator, using the DOM 2 event model

References

1. Programming the World Wide Web, 8th edition
2. Fundamentals of Web Development, 2nd edition
3. https://www.w3schools.com/