Overview

- The **Document Object Model** (DOM) is a programming interface for HTML or XML documents.
- Models document as a tree of nodes.
- Nodes can contain text and other nodes.
- Nodes can have attributes which include style and behavior attributes.
- Possible to get all nodes of a particular type, specific `class` or `id`.

Document Object Model

- API to access parsed HTML/XML documents.
- Can be used from any language, but within browsers the only language commonly supported currently is JavaScript.
- Datatypes include `document`, `element`, `attribute`.
- Global element is `window`.
- All properties of `window` object also available as global variables.

Document

- Current document available as `document` property of global `window` object. Hence available simply as `document`.
- Properties include `location` (URL, giving `href`, `protocol`, `hostname`, `port`, `pathname`, `search`, `hash`), `contentType`, `body`, `cookie` (cookie defs separated by `;`).
- Methods include `getElementsByTagName()`, `getElementsByName()`, `getElementById()`, `getElementsByClassName()`, `querySelector()`, `querySelectorAll()`.
- Allows updating document content dynamically *Dynamic HTML* (DHTML).

Element

- Represents an individual HTML element.
- Properties include `id`, `classList`, `innerHTML` (markup within element), `attributes` (map `NamedNodeMap` of attributes).
- Methods include `getAttribute()`, `getAttributeNames()`, `removeAttribute()`, `setAttribute()`.

Stylesheets

- Current best practice is to relegate presentation to stylesheets.
- Can be specified using **external** stylesheets, using `<link>` elements.
- Can also be specified using **internal** stylesheets using `<style>` elements.
- Can also be specified **inline** for an individual element using `style` attribute.
- Precedence (in descending order) inline, internal, external.

Cascading Style Sheets

- *Cascading Style Sheets* (CSS) specifies priority rules (cascade) between different style declarations which may apply to a element.
- A CSS stylesheet consists of a set of rules.
- A rule consists of a selector followed by a brace delimited set of CSS declarations separated by ;.

```
p .highlight {
 background-color: yellow;
 color: blue
}
```

- Will not cover CSS declarations.

Simple CSS Selectors

Universal Selector	`*` selects all elements; usually used in conjunction with other selectors.
HTML Element Names	Simply specify name of HTML element. Examples `p`, `a`, `table`.
Class Selectors	Name of class preceeded by a `.`. Examples `.highlight`, `.important`.
ID Selectors	ID of element preceeded by `#`. Examples include `#form1`, `#table1`. Note that ID must be unique in document.
`[attr]`	Selects all elements having attribute `attr`. Examples `[href]`,

Combining Selectors

Constrain	Can follow selector by class or id selectors (without spaces). `p.chemical` matches p elements having class `chemical`.
Descendent	Simply write selectors adjacent to each other separated by a space. Example: `.chemical p` selects all p elements which are descendents of a element which has class `chemical`.
Child	Write selectors separated by a >. Example: `.chemical > p` selects all p elements which are direct children of a element which has class `chemical`.

Combining Selectors Continued

Sibling
: Write selectors separated by a `~`. Example: `.chemical ~ p` selects all p elements which follow (not necessarily immediately) a element which has class `chemical`.

Adjacent Sibling
: Write selectors separated by a +. Example: `.chemical + p` selects all p elements which immediately follow a element which has class `chemical`.

Unobstrusive JavaScript

Different technologies used for different concerns:

Content	HTML used for content.
Presentation	CSS used for styling.
Behavior	JavaScript used to specify behavior.

- Do not mix technologies.
- Best practice is to split out into separate `*.html`, `*.css` and `*.js` files.
- Modern technology blurs lines between concerns; CSS 3 contains support for visual behavior traditionally achieved using JavaScript. Nevertheless it remains a good organizational principle.

Bad Code

In `doc.html`:

```
<a href="submit.cgi"
   onClick="checkForm(this)"
   style="font-weight: bold">
  Submit
</a>
```

- Uses CSS and JavaScript code within attributes of HTML elements.
- Maintaining file will require content, presentational and programming skills.

Better Code

1. In `doc.html` maintained by content specialist or a *Content Management System* (CMS):'<a href="submit.cgi" id="submit">Submit</a>'.
2. In `doc.css` maintained by web designer `#submit { font-weight: bold; }`.
3. In `doc.js` maintained by front-end programmer: `document.¬ getElementById('submit').onclick(checkForm(this))`.

- Separate concerns, separate files, separate specialists.
- `doc.html` will need to reference `doc.css` stylesheet and `doc.js`.
- In practice, single `.css` stylesheet, `.js` file shared by multiple html documents.

Playing with the DOM

dom-play.html

Events

- When browser events (like key presses, mouse clicks, page loads) occur, browser calls a **event handler**.
- Historically, different browsers had different ideas of how a event was propagated between an element and its containing elements.
- DOM level 0 allows you to assign a **single** handler to each event for an element using syntax like `element.onclick = function(event) { ... }`. Problematic in that different scripts may each try to add handlers for the same event.
- In DOM level 0 event bubbles up from leaf element on which event occurs to its parent all the way up the DOM tree.
- DOM level 2 event model has a *capture phase* (before *bubble phase*) where event propagated down from the top level of the DOM tree to the leaf element causing the event.
- DOM level 2 allows adding **multiple** handers for an event using `addEventListener(eventType, handler, useCapture)`.

Some DOM Events

MDN

- `DOMContentLoaded`: Initial HTML document loaded and parsed; stylesheets, images, asynchronous scripts may still be loading.
- `load`: complete document, including all dependent resources have been loaded.
- **Focus events** `focus`, `blur`.
- `submit`: a form is being submitted.
- **Keyboard events**: `keydown`, `keyup`, `keypressed`; the last fires continuously.
- **Mouse events** `click`, `dblclick`, `contextmenu`, `mouseenter`, `mouseleave`, `mousemove` (fires continuously), `mouseover`, `mouseout`, `mousedown`, `mouseup`,
- `change`: value of some `<input>`, `<select>` or `<textarea>` element has been changed by the user.

Handler Function

- Within handler function `this` is set to the DOM element on which the handler was registered.
- First argument is an `Event` object with properties like:
  - `target`: DOM element on which the event was dispatched.
  - `type`: Name of event.
  - For keyboard events `key`: value of active key.
  - For mouse events, properties `client[XY]`, `offset[XY]`, `page[XY]`: coordinates of mouse pointer in local coordinates, relative to target node, relative to entire document.
  - `preventDefault()`: calling this function cancels event.
  - `stopPropagation()`: prevents propagation of event.

events play