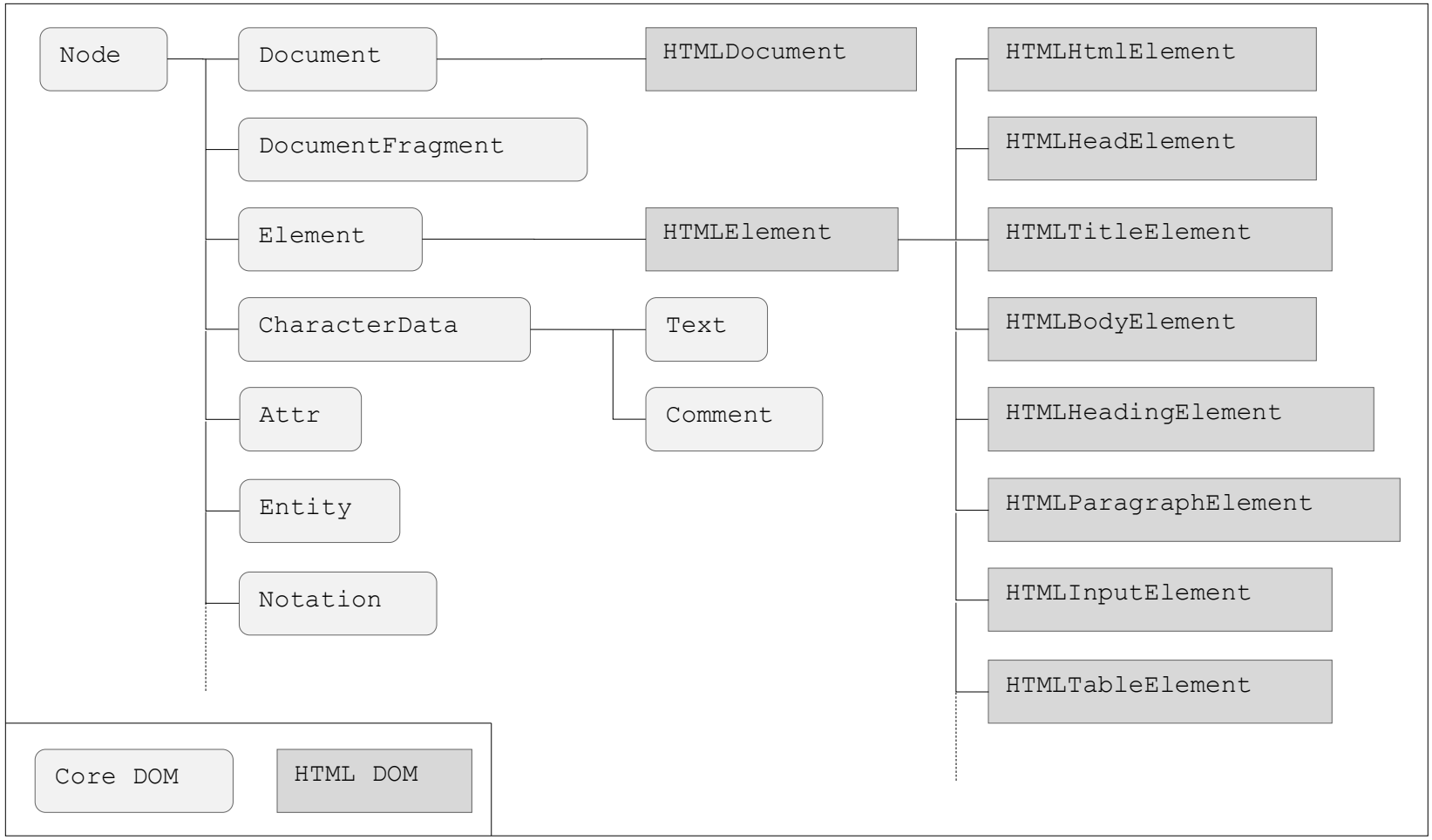
DOM : Document Object Model

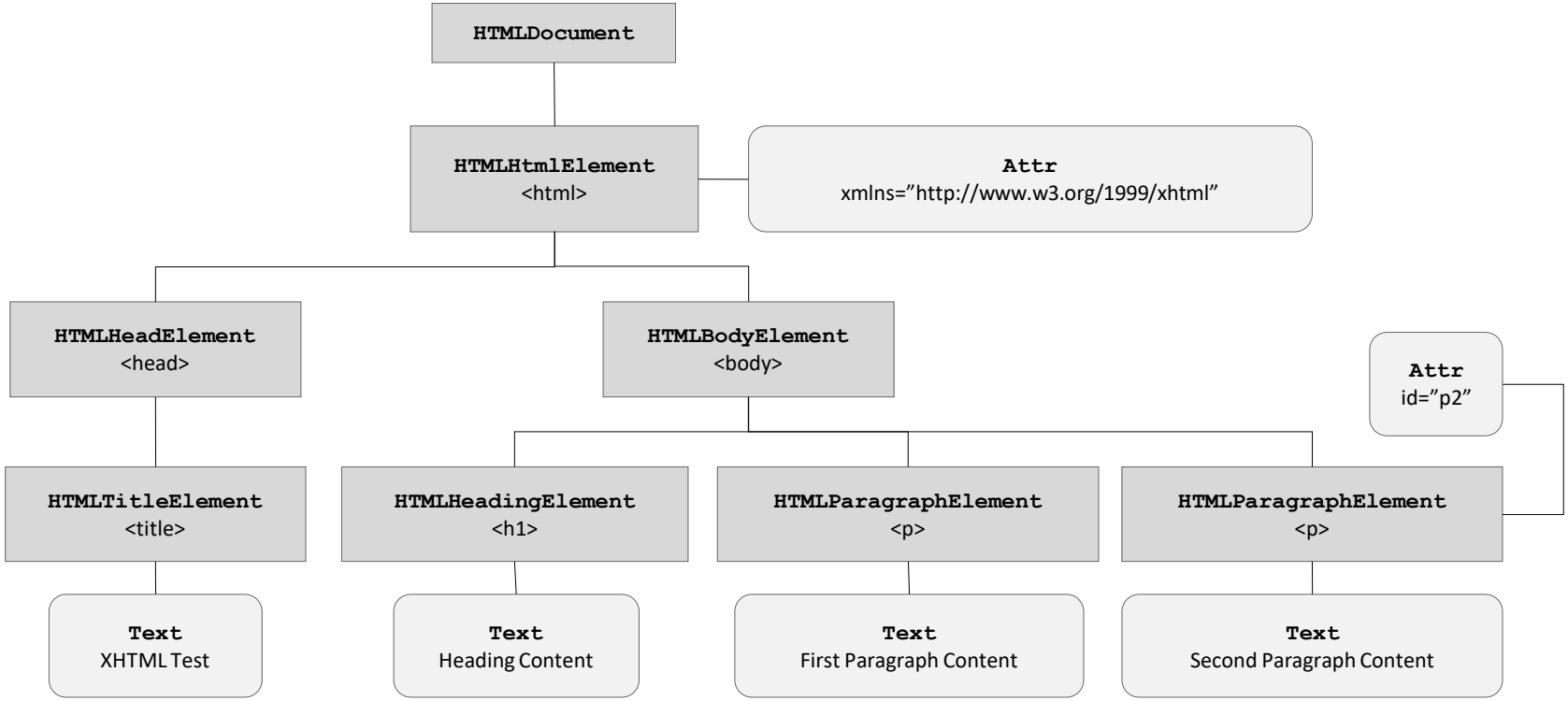
Document Object Model (DOM)

- A standard platform- and language-neutral programming interface for building, accessing, and manipulating valid HTML and well-formed XML documents.
- An interface that must be implemented in an actual programming language to be useful.
- Ultimate goal is to make it possible for programmers to write applications that work properly on all browsers and servers, and on all platforms.
- A tree-based model in which the entire document is parsed and cached in memory as a tree structure of objects called *nodes*.


Node
Document
HTMLDocument
HTMLHtmlElement
DocumentFragment
HTMLHeadElement
Element
HTMLElement
HTMLTitleElement
CharacterData
Text
HTMLBodyElement
Attr
Comment
HTMLHeadingElement
Entity
HTMLParagraphElement
Notation
HTMLInputElement
HTMLTableElement
Core DOM
HTML DOM

- When a Web browser parses an HTML document, it creates an instance of `HTMLDocument`, which encapsulates the entire document – it becomes the root of the tree structure. The Web browser's DOM parser creates objects for every part of the document, all of which implement the `Node` interface. An HTML document is mostly parsed into three basic nodes: `Element` nodes, `Text` nodes, and `Attr` nodes. For example, the following HTML parses into the following tree.

```
<html>
  <head><title>XHTML Test</title></head>
  <body>
    <h1>Heading Content</h1>
    <p>First Paragraph Content</p>
    <p id="p2">Second Paragraph Content</p>
  </body>
</html>
```


HTMLDocument
HTMLHtmlElement
<html>
Attr
xmlns="http://www.w3.org/1999/xhtml"
HTMLHeadElement
<head>
HTMLBodyElement
<body>
Attr
id="p2"
HTMLTitleElement
<title>
HTMLHeadingElement
<h1>
HTMLParagraphElement
<p>
HTMLParagraphElement
<p>
Text
XHTML Test
Text
Heading Content
Text
First Paragraph Content
Text
Second Paragraph Content

- The terminology for the relationships between the nodes is the same as that used for family trees. The node directly above a node in the hierarchy is the *parent* of that node (e.g., `<html>` is the parent of `<body>`). All nodes have exactly one parent, except the *root* node, which has no parent (`HTMLDocument` in the figure above is the root of the HTML hierarchy; however, `<html>` is considered the root of the document that was parsed). The nodes directly below a node are the *children* of that node (e.g., `<head>` and `<html>` are both children of `<html>`). Nodes with no children are called *leaves*. Nodes at the same level and with the same parent are *siblings* (e.g., `<head>` and `<body>` are siblings). All nodes below a node in the hierarchy are the *descendants* of that node. All nodes above a node in the hierarchy are *ancestors* of that node.

DOM Accessors

`Document` interface

`Element documentElement`: The document that was parsed, which is always `<html>` for an HTML document. There is only one document element in an instance of `Document`. The `documentElement` property of the `Document` interface is a convenient reference to the document element. Example:

```
document.documentElement.getAttribute("xmlns");
```

`Element getElementById(DOMString elementId)`: Returns the element with an `id` attribute equal to the given ID or null if no object with that ID exists. You can assign an `id` attribute to any element in an HTML document. The ID you choose must be unique within the HTML document. Example:

```
var secondParagraph = document.getElementById("p2");
```

`NodeList getElementsByTagName(DOMString tagname)`:
Returns a `NodeList` of all elements with the given tag name or an empty list if no tags have the given name. The method also accepts an asterisk ("*") as a wildcard to return all elements.

```
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
    alert(paragraphs.item(i).firstChild.nodeValue);
    // Or use array notation: paragraphs[i].firstChild.nodeValue
}
```

`Element` interface

`DOMString getAttribute(DOMString name)`: Returns the value for the attribute with the given name.

`NodeList getElementsByTagName(DOMString name)`: Similar to the method with the same name in the `Document` interface, except only searches below the `Element`.

Node interface

`DOMString nodeName`

`DOMString nodeValue`: Returns the name of the node and the value of the node. The name of the node is the tag name for nodes representing HTML tags, such as “HEAD” for the `<head>` tag. For other nodes, it’s a value representative of the node as defined by the DOM (e.g., “#text” for `Text` nodes). The value of the node is really only useful for nodes that contain text, like the `Text` node, because for most other nodes the value is `null`.

`NodeList childNodes`: Returns a `NodeList` containing all of the nodes immediate children. The `childNodes` property does not return grandchildren, i.e., a child of a child. Only `Element` nodes have children.

Node parentNode

Node firstChild

Node lastChild

Node previousSibling

Node nextSibling: `parentNode` returns the parent node (only `Element` nodes are capable of being parents, but all nodes, except for `Document`, have a parent). `firstChild` returns the first node in the `NodeList` returned by `childNodes`, while `lastChild` returns the last node in the list. When two nodes have the same parent they are called siblings, which means both the nodes appear in the `NodeList` returned by the parent's `childNodes` property. `previousSibling` returns the sibling node that comes before it in the `childNodes` list, while `nextSibling` returns the node that comes after it.

DOM Modifiers

`Document` interface

`Element createElement(DOMString tagName)`: Creates an `Element` of the given type. The given `tagName` is the name of the element you wish to create. The new node must be added to an document using the `Node` methods. Example:

```
var hrElement = document.createElement("hr");
```

`Text createTextNode(DOMString data)`: Creates a new `Text` node containing the given text. The new node must be added to an document using the `Node` methods. Example:

```
var paragraphText =
  document.createTextNode("Third Paragraph Content");
```

`Node importNode(Node importedNode, boolean deep)`: Imports a node from another document to this document without modifying the other document.

`importNode()` example:

```
var otherParagraph =
  window.parent.frames[1].getElementsByTagName("p")[0];
var newParagraph =
  document.importNode(otherParagraph, true);
```

`Element` interface

`void setAttribute(DOMString name, DOMString value)`: Adds an attribute to the `Element` or, if the attribute already exists, sets the attribute to the given value. Example:

```
document.getElementsByTagName("p")[0].setAttribute("id",
"p1");
```

`void removeAttribute(DOMString name)`: Removes the attribute with the given name. If the attribute doesn't exist, then the method has no effect.

Node interface

`Node insertBefore(Node newChild, Node refChild):`

Inserts the new child in the `childNodes` list before the given existing child and returns the inserted node. Example:

```
var newParagraph = document.createElement("p");
var firstParagraph = document.getElementsByTagName("p")[0];
document.body.insertBefore(newParagraph, firstParagraph);
```

`Node replaceChild(Node newChild, Node oldChild):`

Replaces an existing child with a new child and returns the old child node. Example:

```
var hRule = document.createElement("hr");
var firstParagraph = document.getElementsByTagName("p")[0];
document.body.replaceChild(hRule, firstParagraph);
```

`Node removeChild(Node oldChild)`: Removes the specified child and returns it. Example:

```
var firstParagraph =
  document.getElementsByTagName("p")[0];
document.body.removeChild(firstParagraph);
```

`Node appendChild(Node newChild)`: Adds the given new child to the end of the childNodes list and returns the newly added node. Example:

```
var newParagraph = document.createElement("p");
document.body.appendChild(newParagarph);
```

Manipulating Styles

- The DOM exposes imported style sheets as instances of the `CSSStyleSheet` interface. Each style sheet object exposes the style rules, which you can inspect and modify. You access the style sheet objects via the `document.styleSheets` array.
- However, the more common method of dynamically modifying style is via the `Element` interface. The DOM exposes the styles for each `Element` via the `Element`'s `style` and `className` properties.
- The `style` property returns an instance of the `CSSStyleDeclaration` interface, which contains properties of all the style attributes supported by the browser's DOM

- For example, the following code sets the color of the document's first paragraph to red.

```
var firstParagraph = document.getElementsByTagName("p")[0];
firstParagraph.style.setProperty("color", "#FF0000", "");
```

- The common way to access and set the style properties of an `Element` is via the `CSS2Properties` interface, which contains a publicly accessible property for every style attribute defined by the DOM. In most modern browsers the object returned by the `style` property implements `CSS2Properties`. For example:

```
var firstParagraph = document.getElementsByTagName("p")[0];
firstParagraph.style.color = "#FF0000";
```

- When you need to change several styles for an `Element` at the same time it is more convenient to define the rules in a style sheet for a class selector, and then use JavaScript to set the selector assigned to the `Element`'s `className` property. For example, if the following styles were defined,

```
.normal { color: #000000; font-style: normal; }
.styled { color: #FF0000; font-style: italic; }
```

then we could use the following code to switch the styles.

```
function toggleStyles() {
  var p2 = document.getElementById("p2");
  p2.className = (p2.className == "normal")
        ? "styled" : "normal";
}
```

Navigator Object

- The navigator object contains information about the browser.

Navigator Object Properties

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
language	Returns the language of the browser
onLine	Determines whether the browser is online
platform	Returns for which platform the browser is compiled
product	Returns the engine name of the browser
userAgent	Returns the user-agent header sent by the browser to the server

Navigator Object Methods

Method	Description
javaEnabled()	Specifies whether or not the browser has Java enabled
taintEnabled()	Removed in JavaScript version 1.2. Specifies whether the browser has data tainting enabled