JS

Eman Fathi

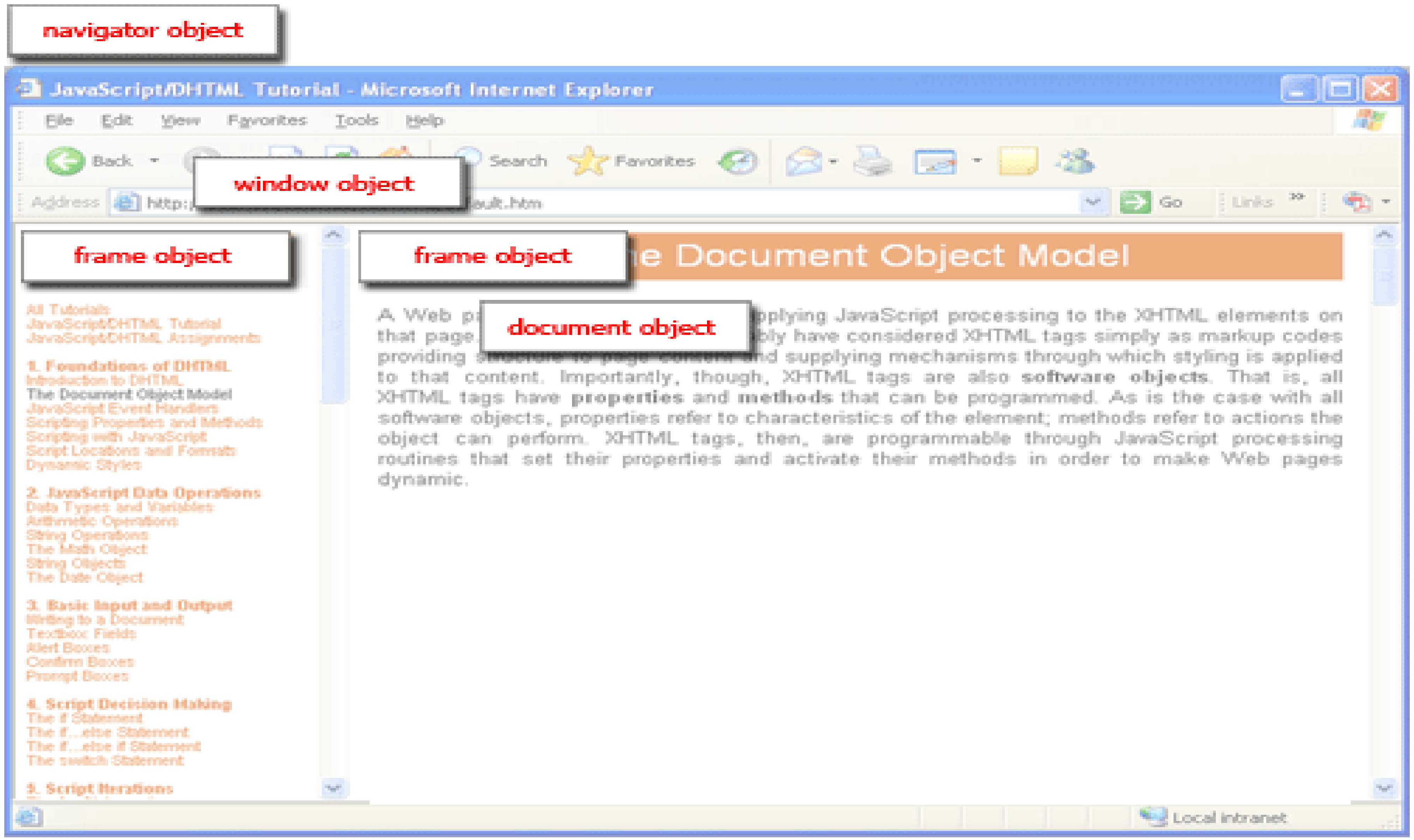
DOM

Document Object Model

What is DOM?

- Document Object Model is the fundamental **API** for **representing** and **manipulating** the content of **HTML** and **XML** documents.
- W3C standard.
- Defines a standard way to access and manipulate HTML documents.
- Allows programmers generic access - **adding**, **deleting**, and **manipulating** - of all styles, **attributes**, and **elements** in a document.
- Platform independent.
- Language independent.

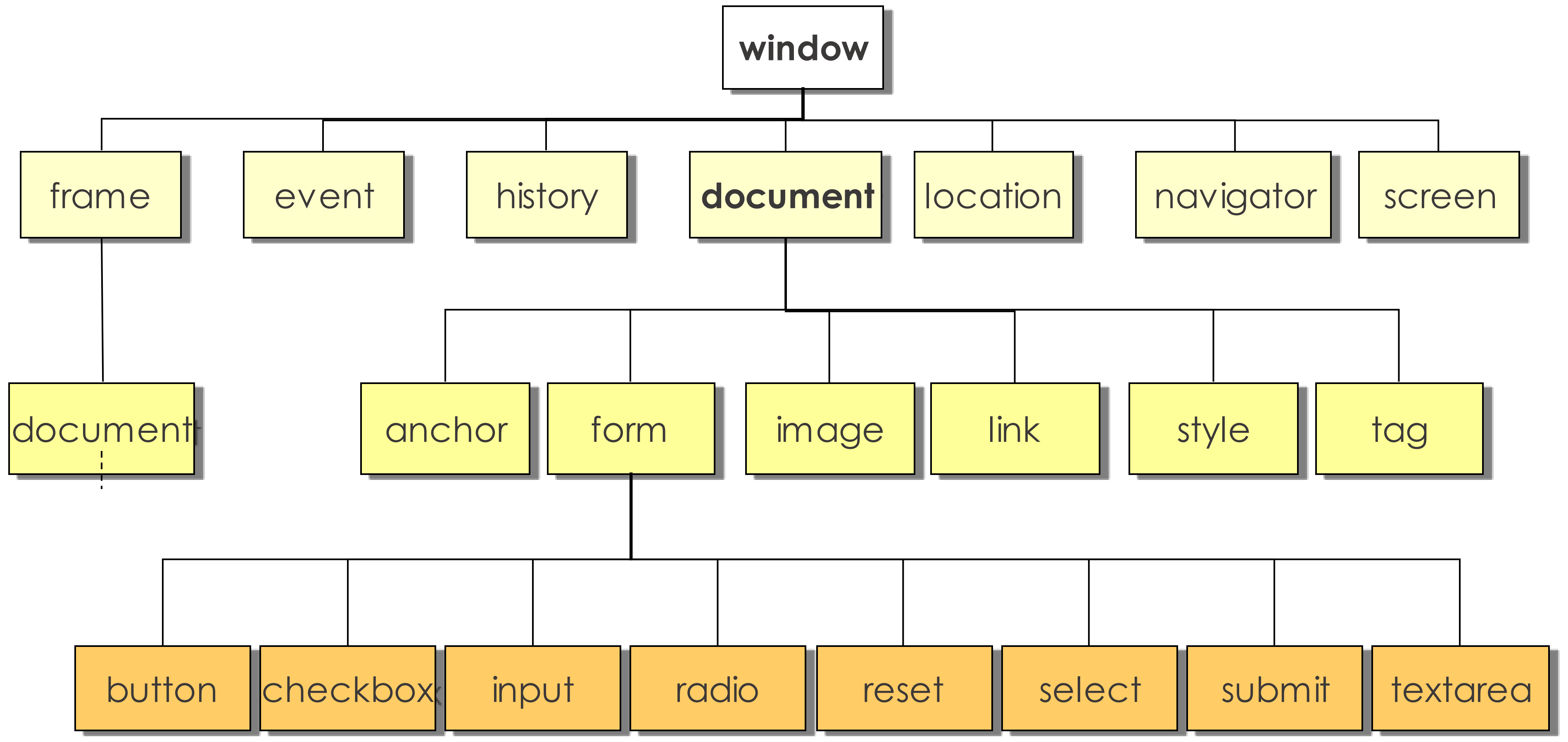
The DOM Model



e Document Object Model

A Web p pplying JavaScript processing to the XHTML elements on that page. bly have considered XHTML tags simply as markup codes providing nd supplying mechanisms through which styling is applied to that content. Importantly, though, XHTML tags are also **software objects**. That is, all XHTML tags have **properties** and **methods** that can be programmed. As is the case with all software objects, properties refer to characteristics of the element; methods refer to actions the object can perform. XHTML tags, then, are programmable through JavaScript processing routines that set their properties and activate their methods in order to make Web pages dynamic.

DOM Tree


window
frame
event
history
document
location
navigator
screen
document
anchor
form
image
link
style
tag
button
checkbox
input
radio
reset
select
submit
textarea

JavaScript Objects Hierarchy

Every page has the following objects:

- **window**: the top-level object; has properties that apply to the entire window.
- **navigator**: has properties related to the name and version of the Navigator being used.
- **document**: contains properties based on the content of the document, such as title, background color, links, and forms.
- **location**: has properties based on the current URL.
- **history**: contains properties representing URLs the client has previously requested.

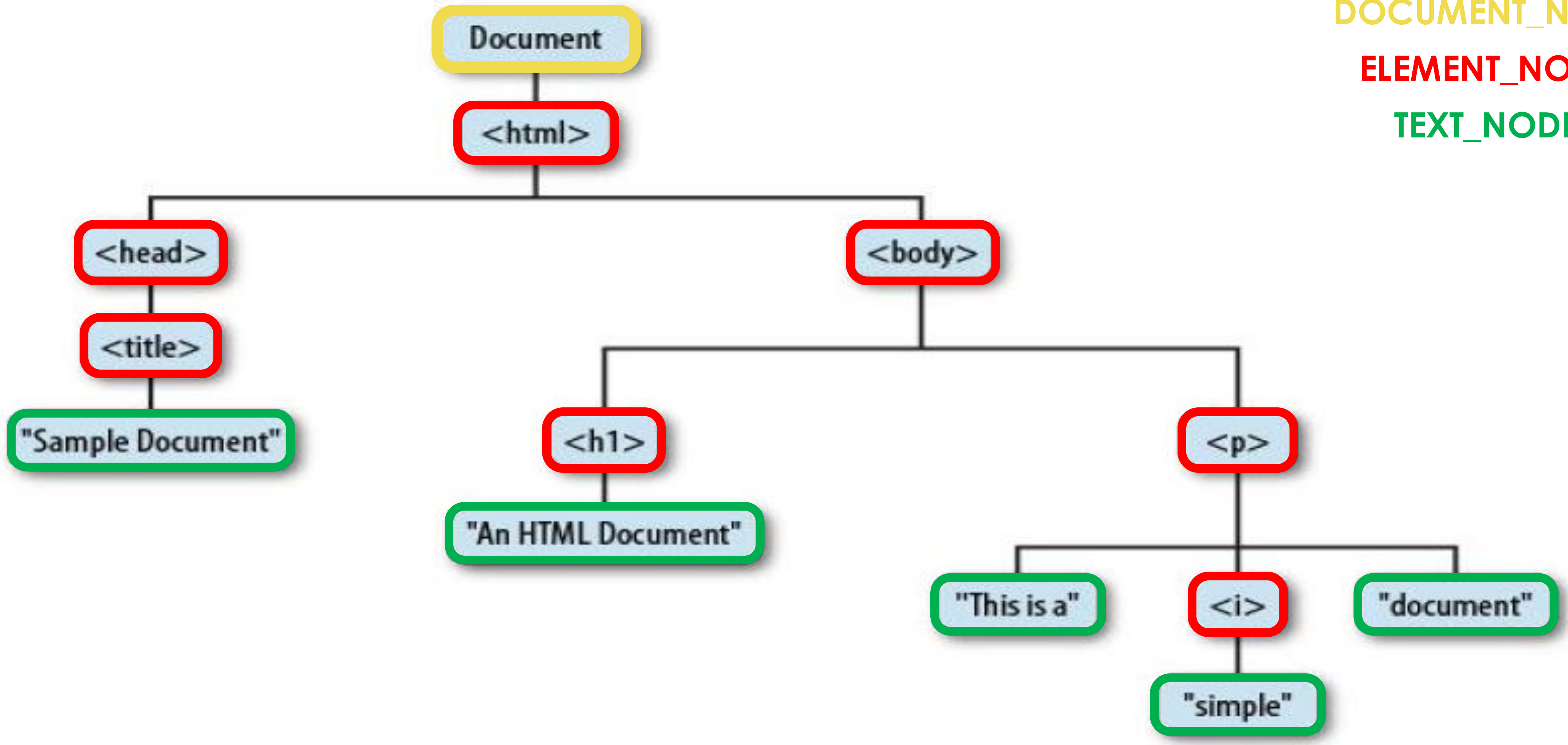
DOM Tree

Nested elements of an HTML or XML document are represented in the DOM as a **tree of nodes**.

The tree representation of an HTML document contains:

- **Nodes representing HTML tags** or elements, such as <body> and <p>,
- **Nodes representing strings of text**.
- **Nodes representing HTML comments.**

```
<html>
    <head>
        <title>Simple Document</title>
    </head>
    <body>
        <h1>Header One</h1>
        <p>This is a <i>simple</i> document</p>
    </body>
</html>
```



DOCUMENT_NODE

ELEMENT_NODE

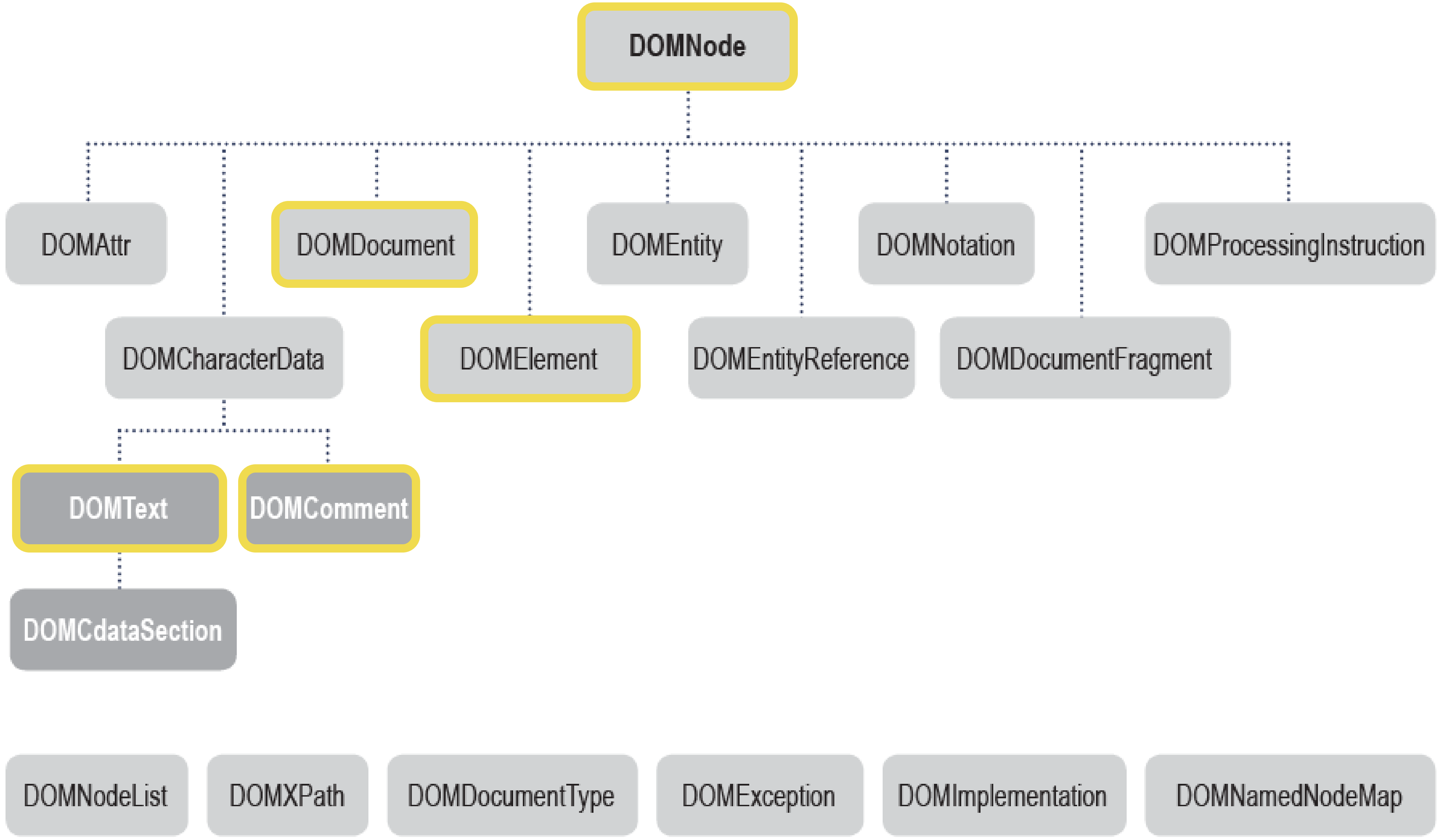
TEXT_NODE

Node

- A Node is an **interface** from which a number of **DOM types inherit**, and allows these various types to be treated (or tested) similarly.

- The following interfaces all inherit from Node its methods and properties: **Document**, **Element**, **CharacterData** (which Text, **Comment**, and **CDATASection** inherit), **ProcessingInstruction**, **DocumentFragment**, **DocumentType**, **Notation**, **Entity**, **EntityReference**

```
Node.ownerDocument Read only

//Returns the Document that this node belongs to. If no document is associated, returns null.

Node.rootNode Read only

//Returns a Node object representing the topmost node in the tree, or the current node if it's
the topmost node in the tree.
```


DOMNode
DOMAttr
DOMDocument
DOMEntity
DOMNotation
DOMProcessingInstruction
DOMCharacterData
DOMElement
DOMEntityReference
DOMDocumentFragment
DOMText
DOMComment
DOMCdataSection
DOMNodeList
DOMXPath
DOMDocumentType
DOMException
DOMImplementation
DOMNamedNodeMap

NodeType

```
Node.nodeType Read only
//Returns an unsigned short representing the type of the node. Possible values are:
```

Name	Value
ELEMENT_NODE	**1**
ATTRIBUTE_NODE	2
TEXT_NODE	**3**
CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	**8**
DOCUMENT_NODE	**9**
DOCUMENT_TYPE_NODE	10
DOCUMENT_FRAGMENT_NODE	11
NOTATION_NODE	12

Node Properties

```
Node.childNodes Read only

//Returns a live NodeList containing all the children of this node. NodeList being live means
that if the children of the Node change, the NodeList object is automatically updated.

Node.firstChild Read only

//Returns a Node representing the first direct child node, or null if the node has no child.

Node.lastChild Read only

//Returns a Node representing the last direct child node, or null if the node has no child.

Node.nextSibling Read only

//Returns a Node representing the next node in the tree, or null if there isn't such node.

Node.parentNode Read only

//Returns a Node that is the parent of this node. If there is no such node, like if this node is
the top of the tree or if doesn't participate in a tree, this property returns null.

Node.parentElement Read only

//Returns an Element that is the parent of this node. If the node has no parent, or if that
parent is not an Element, this property returns null.

Node.previousSibling Read only

//Returns a Node representing the previous node in the tree, or null if there isn't such node.

Node.textContent

//Returns / Sets the textual content of an element and all its descendants.
```

Node Methods

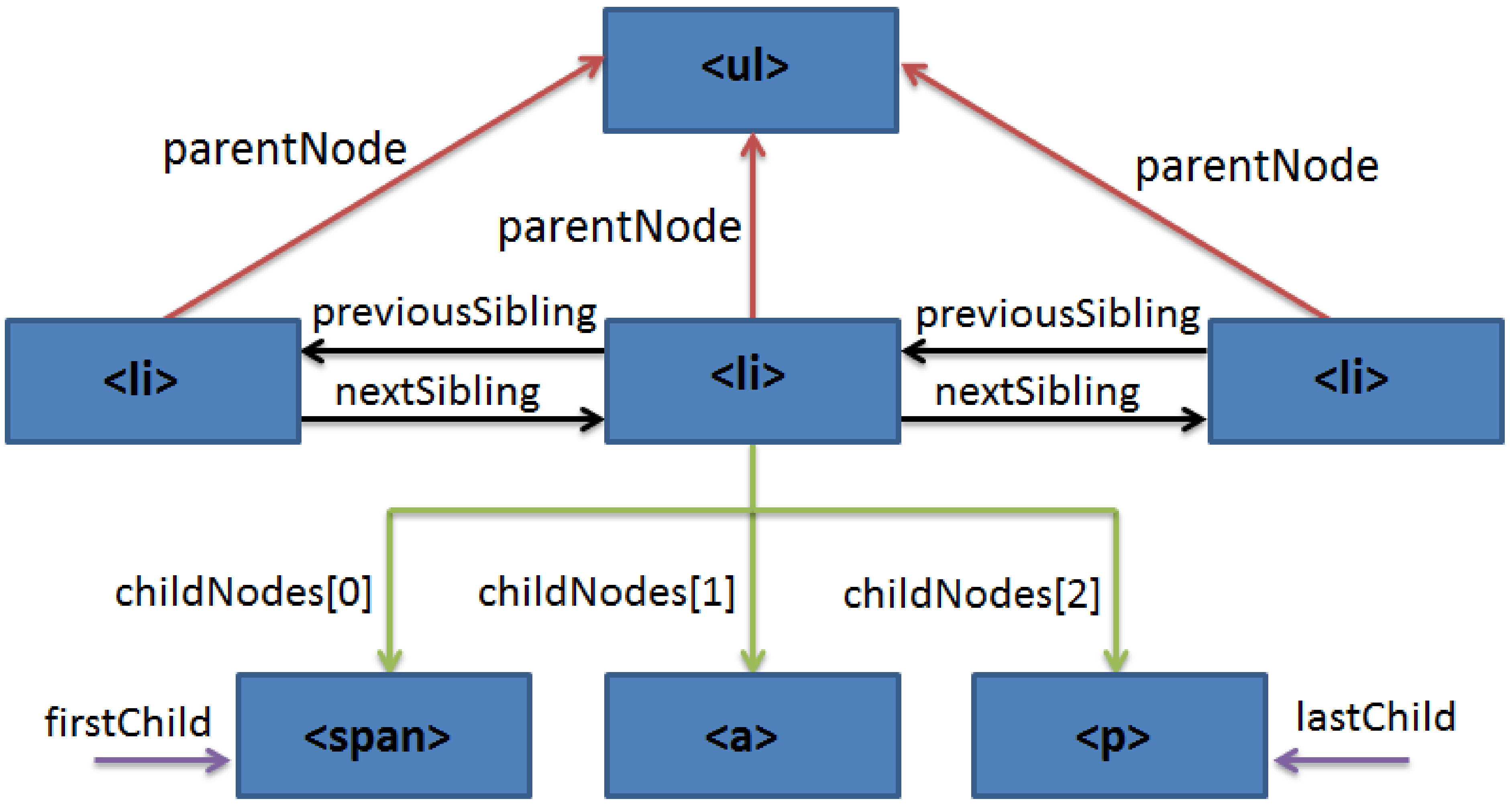
```
Node.appendChild()

//Adds the specified childNode argument as the last child to the current node.

//If the argument referenced an existing node on the DOM gree, the node will be detached from its
current position and attached at the new position.

Node.cloneNode()

//Clone a Node, and optionally, all of its contents.it clones the content of the node by default.

Node.contains()

//Returns a Boolean value indicating whether a node is a descendant of a given node or not.

Node.hasAttributes()

//Returns a Boolean indicating if the element has any attributes, or not.

Node.hasChildNodes()

//Returns a Boolean indicating if the element has any child nodes, or not.

Node.insertBefore()

//Inserts the first Node given in a parameter immediately before the second, child of this
element, Node.

Node.removeChild()

//Removes a child node from the current element, which must be a child of the current node.

Node.replaceChild()

//Replaces one child Node of the current one with the second one given in parameter.
```

Document Structure and Traversal

Document Structure and Traversal

Dealing with document as

- Tree of **Nodes**
- Tree of **Elements**



<ul>
parentNode
parentNode
parentNode
<li>
previousSibling
nextSibling
<li>
previousSibling
nextSibling
<li>
childNodes[0]
childNodes[1]
childNodes[2]
firstChild
<span>
<a>
<p>
lastChild

Document As Trees of Nodes

The Document object, its **Element objects**, and the **Text objects** that represent runs of text in the document are all Node objects.

Node defines the following important properties:

- **parentNode**

The Node that is the parent of this one, or null for nodes like the Document object that have no parent.

- **childNodes**

A read-only array-like object (a NodeList) that is a live representation of a Node's child nodes.

- **firstChild, lastChild**

The first and last child nodes of a node, or null if the node has no children.

Document As Trees of Nodes

- **nextSibling, previousSibling**

The next and previous sibling node of a node. **Two nodes with the same parent are siblings**. Their order reflects the order in which they appear in the document. These properties connect nodes in a doubly linked list.

- **nodeType**

The kind of node this is. Document nodes have the value 9. Element nodes have the value 1. Text nodes have the value 3. Comments nodes are 8

- **nodeValue**

The textual content of a Text or Comment node.

- **nodeName**

The tag name of an Element, converted to uppercase.

DOM Traversing-childNodes

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <!-- This is comment -->
    <h1> Nodes </h1>
    <p> Para1</p>
    <p> Para2</p>
</body>
</html>
document.childNodes                                   //return [<!DOCTYPE html>, html]
document.childNodes[1].childNodes                     //return [head, body]
document.childNodes[1].childNodes[1].childNodes       //[comment,h1, p, text, p, text]
document.childNodes[1].childNodes[1].childNodes[1].childNodes     //[text]
document.childNodes[1].childNodes[1].childNodes[1].firstChild     //" Nodes "
```

DOM Traversing-childNodes

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <!-- This is comment -->
    <h1> Nodes </h1>
    <p> Para1</p>
    <p> Para2</p>
</body>
</html>
document.childNodes[1].lastChild                  //<body>…</body>
document.childNodes[1].lastChild.childNodes[1]   //<h1> Nodes </h1>
document.childNodes[1].lastChild.childNodes[1].nextSibling       //<p> Para1</p>
document.childNodes[1].lastChild.childNodes[1].previousSibling    //<!-- This is comment -->
```

Documents As Trees of Elements

When we are primarily interested in the Elements of a document instead of the text within them (and the whitespace between them), it is helpful to use an API that allows us to treat a document as a tree of Element objects, ignoring Text and Comment nodes that are also part of the document.

- **Children**

Return children

- **firstElementChild, lastElementChild**

Like firstChild and lastChild, but for Element children only.

- **nextElementSibling, previousElementSibling**

Like nextSibling and previousSibling, but for Element siblings only.

- **childElementCount**

The number of element children. Returns the same value as children.length.

DOM Traversing-children

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Traversing</title>
</head>
<body>
    <!-- This is comment -->
    <h1> Nodes </h1>
    <p> Para1</p>
    <p> Para2</p>
</body>
</html>
document.children                                //[html]
document.children[0].children                    //[head, body]
document.children[0].lastElementChild            //<body>…</body>
document.children[0].lastElementChild.children       //[h1, p, p]
document.children[0].lastElementChild.children[1].nextElementSibling     //<p> Para2</p>
document.children[0].lastElementChild.children[1].previousElementSibling //<h1> Nodes </h1>
```

Document

Document Object

- The **central object in DOM** for representing and manipulating document content.

- The Document interface represents any **web page** loaded in the browser and serves as an **entry point into the web page's content**, which is the DOM tree. The DOM tree includes elements such as <body> and <table>, among many others. It provides **functionality** global to the document, like how to obtain the page's URL and **create new elements** in the document.

Document Properties

```
Document.title
//Sets or gets title of the current document.
Document.bgColor
//Gets/sets the background color of the current document.
document.fgColor
//Gets/sets the foreground color, or text color, of the current document.
Document.width
//Returns the width of the current document.
Document.height
//Gets/sets the height of the current document.
Document.linkColor
//Gets/sets the color of hyperlinks in the document.
Document.alinkColor
//Returns or sets the color of active links in the document body.
Document.vlinkColor
//Gets/sets the color of visited hyperlinks.
Document.head Read only
//Returns the <head> element of the current document.
```

Document Properties

```
Document.body
//Returns the <body> element of the current document.
Document.activeElement Read only
//Returns the currently focused element.
Document.cookie
//Returns a semicolon-separated list of the cookies for that document or sets a single cookie.
Document.dir Read only
//Gets/sets directionality (rtl/ltr) of the document.
Document.domain Read only
//Returns the domain of the current document.
Document.lastModified Read only
//Returns the date on which the document was last modified.
Document.readyState Read only
//Returns loading status of the document.
Document.referrer Read only
//Returns the URI of the page that linked to this page.
Document.URL Read only
//Returns the document location as a string.
```

Document Collections

```
Document.all

//Provides access to all elements with an id. This is a legacy, should use the
document.getElementById() method instead.

Document.anchors

//Returns a list of all of the anchors in the document.

Document.embeds Read only

//Returns a list of the embedded <embed> elements within the current document.

Document.forms Read only

//Returns a list of the <form> elements within the current document.

Document.images Read only

//Returns a list of the images in the current document.

Document.links Read only

//Returns a list of all the hyperlinks in the document.

Document.plugins Read only

//Returns a list of the available plugins.

Document.scripts Read only

//Returns all the <script> elements on the document.

Document.styleSheets Read only

//Returns a list of the style sheet objects on the current document.
```

Document Methods

```
document.write(String text)
//Writes text in a document.
document.writeln(String text)
//Writes a line of text in a document.

document.getElementById(String id)
//Returns an object reference to the identified element.
document.getElementsByName(String name)
//Returns a list of elements with the given name.
Document.getElementsByClassName()
//Returns a list of elements with the given class name.
Document.getElementsByTagName()
//Returns a list of elements with the given tag name.
document.querySelector(String selector)
//Returns the first Element node within the document, that matches the specified selectors.
document.querySelectorAll(String selector)
//Returns a list of all the Element nodes within the document that match the specified selectors.
```

Document Methods

```
Document.createAttribute()
//Creates a new Attr object and returns it.
Document.createAttributeNS()
//Creates a new attribute node in a given namespace and returns it.
Document.createCDATASection()
//Creates a new CDATA node and returns it.
Document.createComment()
//Creates a new comment node and returns it.
Document.createElement()
//Creates a new element with the given tag name.
Document.createElementNS()
//Creates a new element with the given tag name and namespace URI.
Document.createEvent()
//Creates an event object.
Document.createTextNode()
//Creates a text node.
Document.createTouch()
//Creates a Touch object.
```

Element

- The Element interface represents an object of a Document.
- This interface describes methods and properties common to all kinds of elements. Specific behaviors are described in interfaces which inherit from Element but add additional functionality. For example, the **HTMLElement** interface is the base interface for HTML elements, while the **SVGElement** interface is the basis for all SVG elements.

Element Properties

```
Element.attributes Read only
//Returns a NamedNodeMap object that cotaining the assigned attributes of the HTML element.
Element.classList Read only
//Returns a DOMTokenList containing the list of class attributes.
Element.className
//Is a DOMString representing the class of the element.
Element.clientHeight Read only
//Returns a Number representing the inner height of the element.
Element.clientLeft Read only
//Returns a Number representing the width of the left border of the element.
Element.clientTop Read only
//Returns a Number representing the width of the top border of the element.
Element.clientWidth Read only
//Returns a Number representing the inner width of the element.
Element.id
//Is a DOMString representing the id of the element.
Element.tagName Read only
//Returns a String with the name of the tag for the given element.
```

```
Element.innerHTML

//Is a DOMString representing the markup of the element's content.

Element.outerHTML

//Is a DOMString representing the markup of the element including its content. When used as a
setter, replaces the element with nodes parsed from the given string.

Element.scrollHeight Read only

//Returns a Number representing the scroll view height of an element.

Element.scrollLeft

//Is a Number representing the left scroll offset of the element.

Element.scrollLeftMax Read only

//Returns a Number representing the maximum left scroll offset possible for the element.

Element.scrollTop

//Is a Number representing the top scroll offset the an element.

Element.scrollTopMax Read only

//Returns a Number representing the maximum top scroll offset possible for the element.

Element.scrollWidth Read only

//Returns a Number representing the scroll view width of the element.
```

Find something

Do something

Selecting Document Element

- with a specified **id** attribute
- with a specified **name** attribute
- with the specified **tag** name
- with the specified **CSS class** or classes
- Matching the specified **CSS selector**

Selecting Element By Id

- Any HTML element can have an id attribute.
- The value of this attribute must be **unique** within the document—no two elements in the same document can have the same ID.
- You can select an element based on this unique ID with the getElementById() method of the Document object.

```
      document.getElementById("ElemetnID")
```

HTML code:

```
<div id="container"><span><p> Selecting Elements By Id’s </p></span></div>
```

JS Code:

```
//ID is case-senstive
var contDiv = document.getElementById('container');
```

Selecting Element By Id

There is a single element on page has unique ID, then the value of ID **automatically create window property of element itself**.

Call the element either by getElementById or window property as follow:

```
var contDiv = window.container
//Calling is case-senstive
```

Selecting Element By Name

- Like the id attribute, name assigns a name to an element.
- Unlike id, however, the value of a name attribute does not have to be unique: **multiple elements may have the same name**, and this is common in the case of **radio buttons** and **checkboxes** in forms
- To select HTML elements based on the value of their name attributes, you can use the getElementsByName() method of the Document object

```
document.getElementsByName("ElemetnsName");
```

Selecting Element By Name

```
<form id="UserData" action="Data.aspx" name= "userForm" >
        <input type="checkbox" name="hoppy" value="1" />Running
        <input type="checkbox" name="hoppy" value="2" />Swimming
        <input type="checkbox" name="hoppy" value="3" />Reading
        <input  type="submit" value="Send"/>
</form>
//Name is case-senstive
var userHoppies = document.getElementsByName("hoppy");
```

If there is only a single element with a given name, the value of the element automatically created document or window property is the element itself. If there is more than one element, then the value of the property is a **NodeList** object that acts as an array of elements.

```
var formElm = document.userForm;         //return form tag
var radBtns = document.userForm.hoppy    //return radio buttons
var radBtns = userForm.hoppy
```

Selecting Element By Type

- You can select all HTML or XML elements of a specified type (or tag name) using the getElementsByTagName()

```
var spans = document.getElementsByTagName("span");
//Tag name is Case-Insenstive
```

- Like getElementsByName(), getElementsByTagName() returns a **NodeList** object.
- The elements of the returned NodeList are **in document order**, so you can select the first <p> element of a document like this:

```
var firstpara = document.getElementsByTagName("p")[0];
```

Selecting Element By Type

- You can obtain a NodeList that represents all elements in a document by passing the wildcard argument "*" to the method

```
 var allTags=document.getElementsByTagName("*");
```

- You can selects elements that are **descendants** of the element on which it is invoked. So to find all <span> elements inside the first <p> element of a document, you could write:

```
var firstpara = document.getElementsByTagName("p")[0];
var firstParaSpans = firstpara.getElementsByTagName("span");
```

Selecting Element By Type

```
<table id="subjects">
        <tr><td> HTML </td><td> CSS </td> </tr>
        <tr><td> Javascript </td><td> AngularJS</td> </tr> </table>
<table id="Instructors">
        <tr><td> Mona </td><td> Ali </td> </tr>
        <tr><td> Noha </td><td> Mohammed</td> </tr>
</table>

//JS Code:
var subjcetTable = document.getElementsByTagName("table")[0]
var subjCells = subjcetTable.getElementsByTagName("td")
```

Selecting Element By Class Name

- The class attribute of an HTML is a **space-separated list of zero or more identifiers**.
- HTML5 defines a method, getElementsByClassName(), that allows us to select sets of document elements based on the identifiers in their class attribute.

```
document.getElementsByClassName("ClassName");
```

- getElementsByClassName() takes a single string argument, but the string may specify multiple space-separated identifiers. **Only elements that include all of the specified identifiers in their class attribute are matched**. (The order of the identifiers does not matter)

```
document.getElementsByClassName("Class1 Class2 …");
```

Selecting Element By Class Name

```html
<div id="container">
    <span id="spanElm1" name="t" class="fontLarge bPink">This is span </span>
    <div name="divElm" class="bPink fontGreen"> This is div </div>
</div>
```

JS Code:

```javascript
var Elms1 = document.getElementsByClassName("bPink"); //returns div and span

var Elms2 = document.getElementsByClassName("fontGreen bPink"); //returns div only

// combine all selection methods to retrieve elements
// Find all children of the element named "container" that have the class "fontGreen"
var conElm = document.getElementById("container");
var elms = conElm.getElementsByClassName("fontGreen");
```

Selecting Element By CSS Selectors

CSS stylesheets have a very powerful syntax, known as ***selectors***, for describing elements or sets of elements within a document.

Elements can be described by ID, tag name, class or attributes:

```
#nav               // An element with id="nav"
div                // Any <div> element
.warning           // Any element with "warning" in its class attribute
p[lang="fr"]       // A paragraph written in French: <p lang="fr">
```

Selecting Element By CSS Selectors

- The key to Selectors API is the Document methods:
  - querySelector()
  - querySelectorAll()

querySelectorAll()

- It takes a single string argument containing a CSS selector and returns a **NodeList** that represents all elements in the document **that match the selector**.
- If no elements match, querySelectorAll() returns an empty NodeList.
- If the selector string is invalid, querySelectorAll() throws an exception.

querySelector()

- **returns only the first** (in document order) **matching element**.
- **returns null** if there is no matching element.

Basic Selectors

```
//1- Tag name
document.querySelectorAll("p");
document.querySelectorAll("p,li");
//2- class name
document.querySelectorAll(".a");
document.querySelectorAll("p.a");
//more than one class (order is not important)
document.querySelectorAll(".bPink.fontGreen")
//3- ID
document.querySelectorAll("#spanElm");
//4- Select all Document Elements
document.querySelectorAll("*")
//grouping
document.querySelectorAll("p,.a,#spanElm,li")
```

Hierarchical Selectors

```
//Direct Child. document.querySelectorAll("div > p")
//select child element that an immediate descendants of the parent )

//Descendants. document.querySelectorAll("div  p")
//select descendants as long as they have ancestor above them


//Next Adjacent. document.querySelectorAll("div + img")
//select the next element if it is immediately preceded By prev element

//Next Siblings. document.querySelectorAll("div ~ img")
//selects all siblings come after prev element
```

Attribute Selectors

```
// Any element has specified Attribute
document.querySelectorAll("p[class]");
document.querySelectorAll("input[type=button]");
// starts with
document.querySelectorAll("a[href^=http]")
//ends with
document.querySelectorAll("a[href$='.com']")
//contains
document.querySelectorAll("a[href*='iti']")
//multiple attributes
document.querySelectorAll("input[type=text][value*='here']")
```

Position Filters

```
//select first element
document.querySelectorAll("li:first-child");

//select last element
document.querySelectorAll("li:last-child");

//select element at specified position (index start with 1)
document.querySelectorAll("li:nth-child(2)");

//odd or even elelments
document.querySelectorAll("tr:nth-child(even)")
document.querySelectorAll("tr:nth-child(odd)")
```

HTML Attributes

HTML Attributes As Element Properties

- The HTMLElement objects that represent the elements of an HTML document define read/write **properties** that **mirror the HTML attributes** of the elements.
- HTML attributes are not case sensitive, but JavaScript property names are.
- To convert an attribute name to the JavaScript property, write it in **lowercase**.

```
<img src="images/bernal.jpg" />


JS Code:
//getting image URL
var image = document.getElementById("myimage");
var imgurl = image.src;  //getter
image.src = "flowers.jpg"; //setter
```

HTML Attributes As Element Properties

- If the attribute is more than one word long, however, put the first letter of each word after the first in uppercase: defaultChecked for example.
- Some HTML attribute names are reserved words in JavaScript. as Class attribute becomes className.
- The properties that represent HTML attributes usually have **string** value.
- When the attribute is a boolean or numeric value (the defaultChecked and maxLength attributes of an <input> element, for example), the properties values are booleans or numbers instead of strings.
- **Event handler** attributes always have **Function** objects (or null) as their values.
- Finally, the value of the style property of any HTML element is a CSSStyleDeclaration **object** rather than a string.

HTML Attributes As Element Properties

```
HTML Code:
<input type="checkbox" id="myCheck" checked>
<input  type="text" class="greenFont"  maxlength="20"/>

JS Code:
document.getElementById("checkBx").defaultChecked        //true

document.querySelector("input[type=text]").className      //string

document.querySelector("input[type=text]").maxLength      //int
```

DOM Element Attribute Methods

Using Element Methods to set and get HTML attributes

- **setAttribute()**
- **getAttribute()**
- **hasAttribute()**
- **removeAttribute()**

Differences between these methods and the element property:

- attribute values are all treated as strings. getAttribute() never returns a number, boolean, or object.
- these methods use standard attribute names, even when those names are reserved words in JavaScript.
- For HTML elements, the attribute names are case insensitive.

setAttribute()

Syntax: `element.setAttribute("attributename","attributevalue")`

Example:

```
var anchorElm = document.getElementById("myAnchor");
anchorElm.setAttribute("href", "http://www.w3schools.com");
anchorElm.setAttribute("class","fontClass");
```

getAttribute()

Syntax: `element.getAttribute("attributename")`

Example:

```
var anchorElm = document.getElementById("myAnchor");
anchorElm.getAttribute("target");
```

hasAttribute()

Syntax: `element.hasAttribute("attributename")`

Example:

```
// If the <a> element has a target attribute, set the value to "_self"
if (x.hasAttribute("target")) {
    x.setAttribute("target", "_self");
}
```

removeAttribute()

Syntax: `element.removeAttribute("attributename")`

Example:

```
anchorElm.removeAttribute("class")
```

Creating, Inserting, and Deleting Nodes

Creating Nodes

createElement() is document object method to create new Elements nodes

Pass the tag name of the element.

Syntax:

```
document.createElement("nodeName");
```

Return Value:

An Element object, which represents the created Element node.

Creating Nodes

createTextNode() is document object method to create new Text Elements nodes

Syntax

```
document.createTextNode("Text Content");
```

Return Value: created Text node

Example:

```
 var node = document.createElement("LI");
// Create a <li> node
var textnode = document.createTextNode("Water");
// Create a text node
```

Inserting Nodes

Once you have a new node, you can insert it into the document with the Node methods:

- appendChild()
- insertBefore()

appendChild()

appendChild() is invoked on the Element node that you want to insert into, and it inserts the specified node so that it becomes the **last Child** of that node.

Method used to append new element or moving element from one element to another.

Syntax

node.appendChild(node)

Return

A Node Object, representing the appended node

appendChild()

Example:

```
//appending Child li to UL
var liItem = document.createElement("li")
var liTxt = document.createTextNode("water")
liItem.appendChild(liTxt);
document.getElementsByTagName("ul")[0].appendChild(liItem);

//moving water to ol
var wLi = document.querySelector("ul").lastChild
document.getElementsByTagName("ol")[0].appendChild(wLi);
```

insertBefore()

insertBefore() is like appendChild(), but it takes **two arguments**:

- The **first argument** is the **node** to be inserted.
- The **second argument** is the **node before which that node IS to be inserted**.
- This **method is invoked on** the node that will be the **parent** of the new node, and the **second argument must be a child of that parent node**.
- If you pass **null** as that second argument, the insertBefore() behaves like appendChild() and inserts at the **end**.

Syntax

node.insertBefore(*newnode,existingnode*)

Return

A Node Object, representing the inserted node

insertBefor()

Example:

```
var newItem = document.createElement("LI");
// //Create a <li> node
var textnode = document.createTextNode("Water");
// //Create a text node
    newItem.appendChild(textnode);                    // Append the text to <li>
var list = document.getElementById("myList");
// Get the <ul> element to insert a new node
list.insertBefore(newItem, list.childNodes[0]);
// Insert <li> before the first child of <ul>
```

clone() Method

Every node has a cloneNode() method that returns a new copy of the node.

Pass **true** to recursively **copy all descendants** as well, or **false** to only make a **shallow copy**.

```
//clone water item without remove it from ul List
document.querySelector("ul").lastElementChild.cloneNode(true)
//return <li>water</li>
document.querySelector("ul").lastElementChild.cloneNode(false)
//return <li></li>
var wLi =document.querySelector("ul").lastChild.cloneNode(true);
document.getElementsByTagName("ol")[0].appendChild(wLi);
```

Removing and Replacing Nodes

- The removeChild() method removes a node from the document tree.
- Be careful, however: this method isn't invoked on the node to be removed on the parent of that node. Invoke the method on the parent node and pass the child node that is to be removed as the method argument.
- replaceChild() removes one child node and replaces it with a new one.
- Invoke this method on the parent node, passing the new node as the first argument and the node to be replaced as the second argument.

removeChild() Method

- The removeChild() method removes a specified child node of the specified element.
- The new node could be an existing node in the document, or you can create a new node.

Syntax

node.removeChild(*node*)

Return

A Node object, representing the removed node, or *null* if the node does not exist

removeChild() Method

Example:

```
// Get the <ul> element with id="myList"
var list = document.getElementById("myList");

// If the <ul> element has any child nodes, remove its first child node
if (list.hasChildNodes())
{
    list.removeChild(list.childNodes[0]);
}
```

replaceChild() Method

- The replaceChild() method replaces a child node with a new node.
- The new node could be an existing node in the document, or you can create a new node.

Syntax

node.replaceChild(*newnode*,*oldnode*)

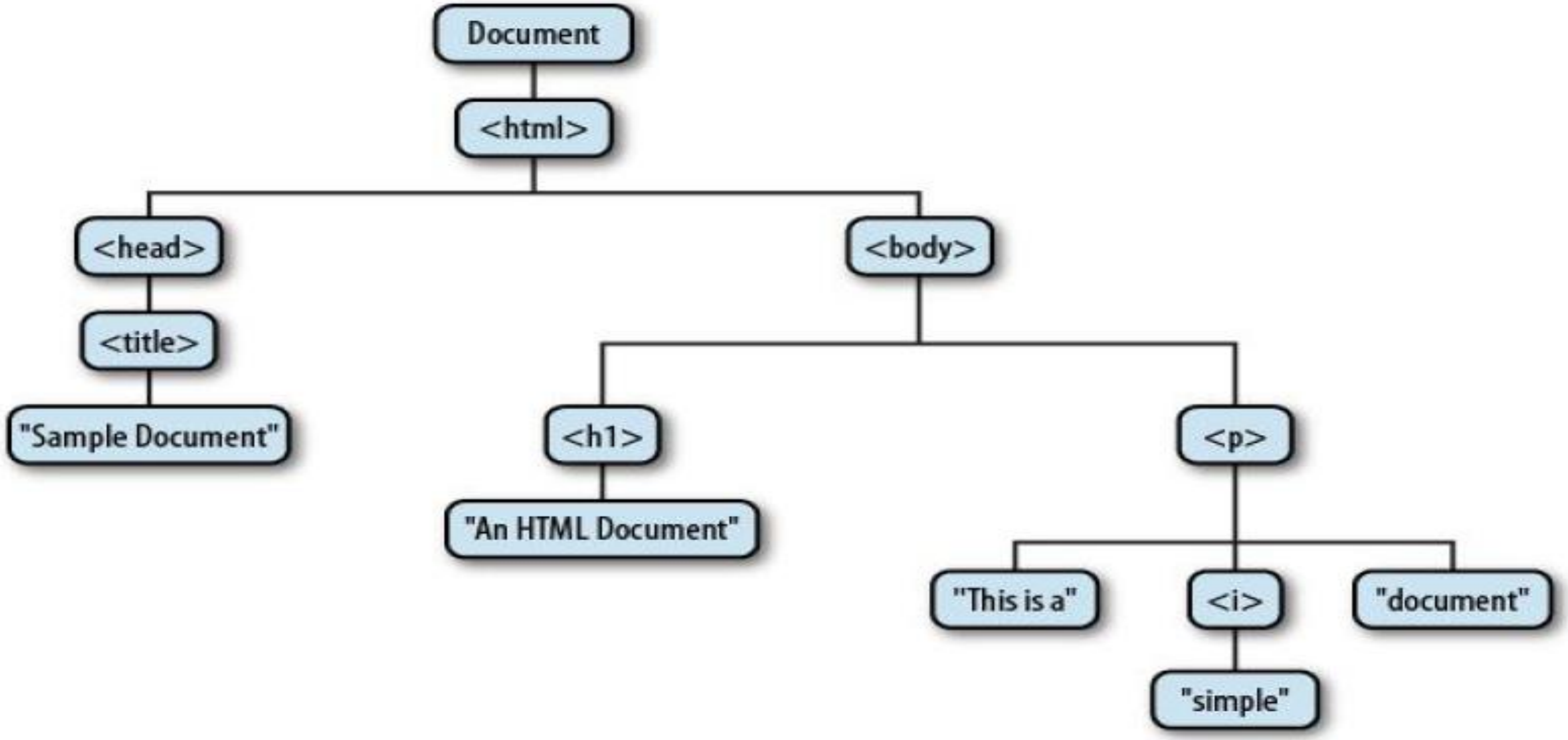
Return

A Node object, representing the replaced node

replaceChild() Method

Example:

```
// Create a new <li> element
var elmnt = document.createElement("li");
var textnode = document.createTextNode("Water");
elmnt.appendChild(textnode);


// Get the <ul> element with id="myList"
var item = document.getElementById("myList");
// Replace the first child node (<li> with index 0) in <ul> with the newly created <li>
element
item.replaceChild(elmnt, item.childNodes[0]);
```



Look at Tree Structure above and ask yourself what the “content” of the <p> element is.

Element Content

There are three ways we might answer this question:

• The content is the HTML string “This is a <i>simple</i> document.”

• The content is the plain-text string “This is a simple document.”

• The content is a Text node, an Element node that has a Text node child, and another Text node.

Element Content As HTML

- Reading the innerHTML property of an Element returns the content of that element as a string of markup. Setting this property on an element invokes the web browser's parser and replaces the element's current content with a parsed representation of the new string.

```
document.querySelector("p").innerHTML; // getter

document.querySelector("p").innerHTML = "Change <b> HTML </b> Contents"; // setter
```

Element Content As Plain Text

- Sometimes you want to query the content of an element as plain text, or to insert plaintext into a document The standard way to do this is with the textContent property of Node:

```
document.querySelector("p").innerText; // getter


document.querySelector("p").innerText = "Change  Text Contents"; // setter
```

Element Content As Text Nodes

- Another way to work with the content of an element is as a list of child nodes, each of which may have its own set of children.

```
//Dealing with Text as Tree of nodes
 document.querySelector("p").childNodes[0].nodeValue


//remember that
document.querySelector("p").children()
//will return null
```

Thank You