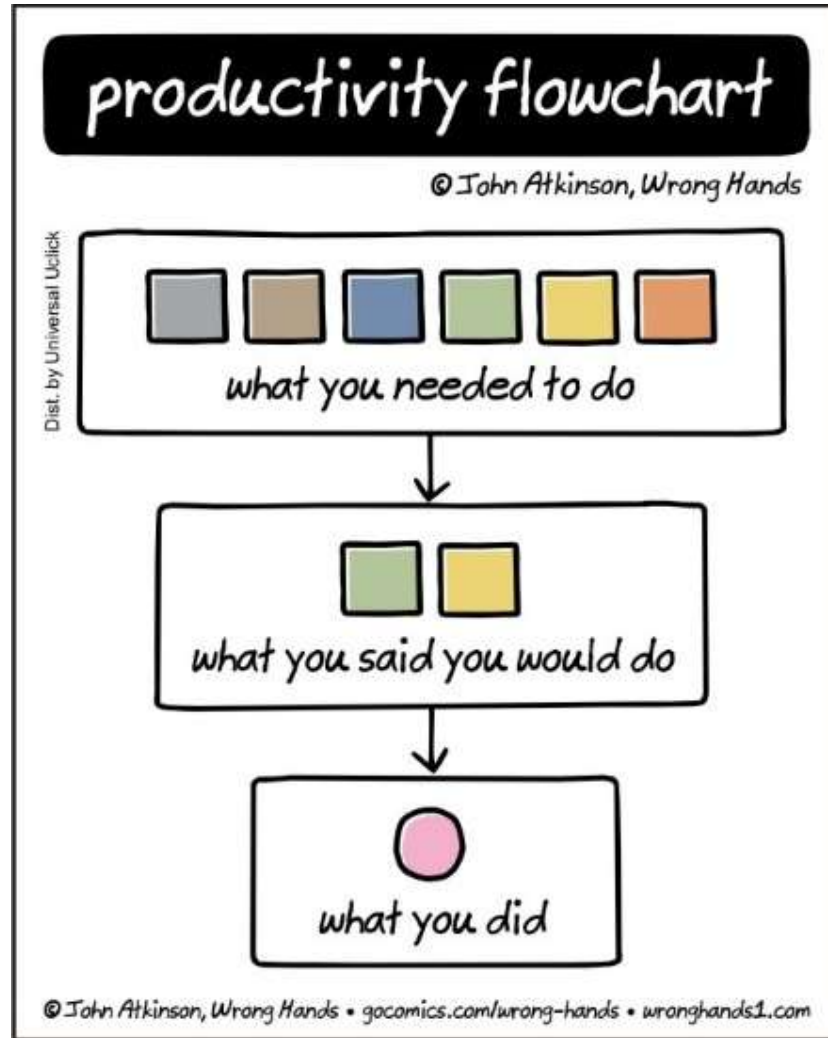


Tehnologii Web



procesarea datelor XML/HTML (I)

DOM – *Document Object Model*

„Regula de aur este că nu există reguli de aur.”

George Bernard Shaw

Cum putem prelucra documentele XML?

Tipuri de procesări XML

procesare manuală

e.g., expresii regulate 😞

Tipuri de procesări XML

procesare obiectuală

DOM (*Document Object Model*)

non-DOM

Tipuri de procesări XML

procesare condusă de evenimente

SAX (Simple API for XML)

XPP (XML Pull Parsing)

vezi cursul viitor

Tipuri de procesări XML

procesare simplificată

Simple XML

vezi cursul viitor

Tipuri de procesări XML

procesare particulară

via API-uri specializate pentru a prelucra
tipuri de documente specifice – *e.g.*, RSS, SOAP, SVG,...

Procesoare (analizoare) XML fără validare

verifică doar dacă documentul
este bine-formatat (*well formed*)

Expat, libxml, MSXML,...

Procesoare (analizoare) XML **cu validare**

verifică dacă documentul este valid,
conform unei metode de validare – *e.g.*, DTD

Apache Xerces, JAXP, libxml, MSXML,...

Modelul DOM

introducere

interfețe DOM

DOM Core

DOM – nivelul 2

DOM – nivelul 3

DOM – nivelul 4

implementări

DOM direct în navigatorul Web

dom: intro

Scop:

procesarea obiectuală – standardizată –
a documentelor XML și/sau HTML

dom: caracterizare

Interfață de programare a aplicațiilor (API)
abstractă pentru XML/HTML

dom: caracterizare

Interfață de programare a aplicațiilor (API)
abstractă pentru XML/HTML

independentă de platformă și limbaj

standardizată de Consorțiul Web

www.w3.org/DOM/DOMTR

dom: caracterizare

Definește o structură logică arborescentă
a documentelor XML

document \equiv ierarhie a unui set de obiecte
pe baza cărora se pot accesa/modifica date XML

dom: niveluri de specificare

DOM 1 (1998)

www.w3.org/TR/REC-DOM-Level-1/

DOM Core pentru XML

DOM HTML pentru procesarea standardizată
a paginilor Web – uzual, la nivel de client (*browser*)

dom: niveluri de specificare

DOM 2 (2001)

www.w3.org/TR/REC-DOM-Level-2/

recomandări multiple privind diverse funcționalități:
spații de nume, aplicare de stiluri,
răspuns la evenimente etc.

dom: niveluri de specificare

DOM 3 (2004)

www.w3.org/TR/DOM-Level-3-Core/

funcționalități specifice oferite de module
(unele deja standardizate)

XPath, traversare a arborelui, validare,
încărcare și salvare (asincrone),...

dom: niveluri de specificare

DOM 4 (2015)

www.w3.org/TR/dom/

restructurarea unor interfețe + noi funcționalități

dom: niveluri de specificare

DOM Living Standard

dom.spec.whatwg.org

specific HTML5

în continuă dezvoltare
(cea mai recentă actualizare: 29 martie 2019)

dom: interfețe

Modalitate abstractă de accesare și de modificare
a reprezentării interne a unui document XML

dom: interfețe

Modalitate abstractă de accesare și de modificare a reprezentării interne a unui document XML

datele sunt încapsulate în obiecte, ascunse și/sau protejate de prelucrarea externă directă

dom: interfețe

Nu implică o implementare concretă, particulară

DOM oferă interfețe
independente de implementare pentru
accesarea/procesarea datelor

dom: interfețe

Interfețele sunt specificate cu
IDL (*Interface Description Language*)

specificația curentă: www.omg.org/spec/IDL/

dom: interfețe – IDL

Definește tipurile de obiecte
prin specificarea interfețelor acestora
(proprietăți + metode publice)

pur declarativ

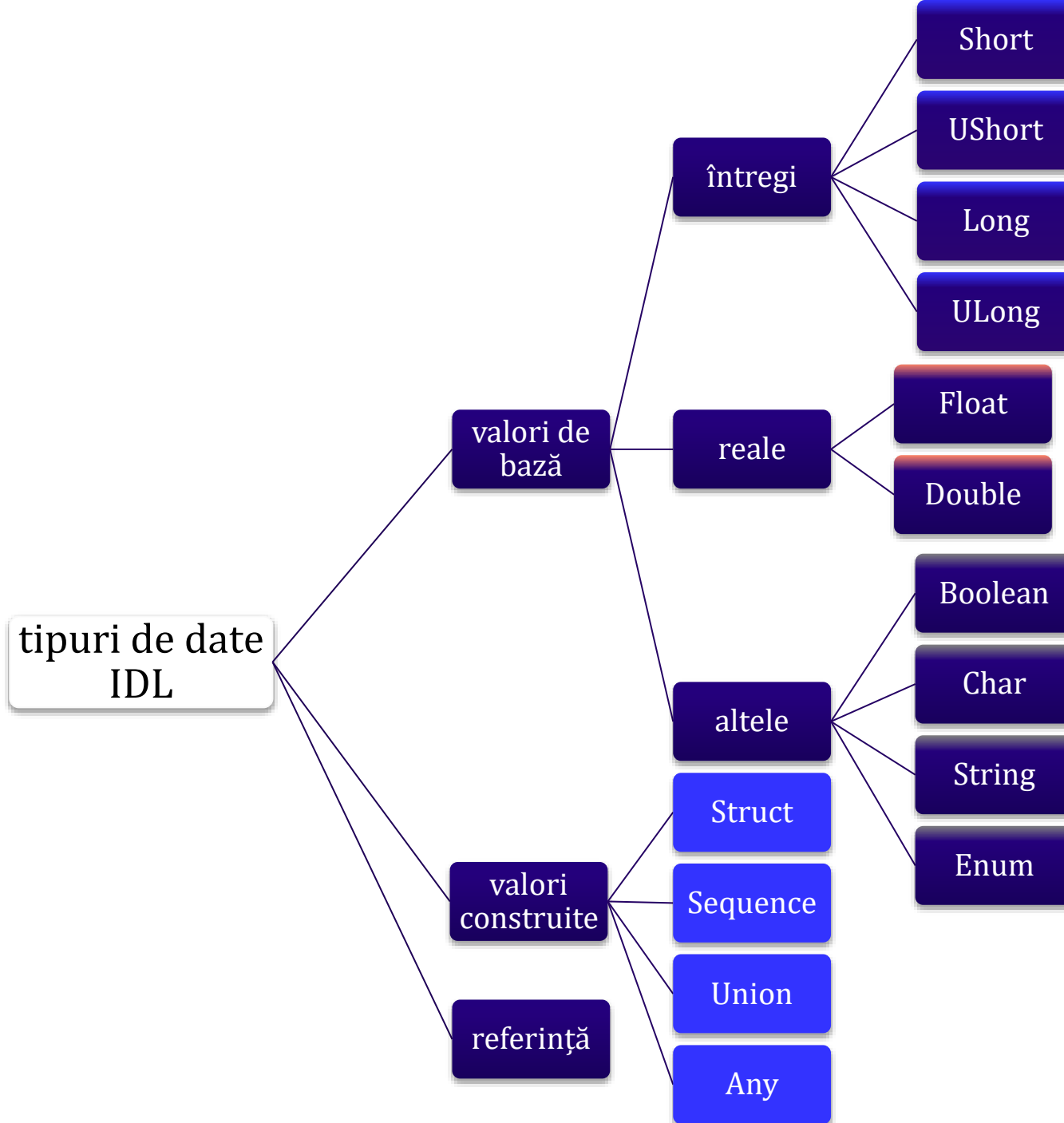
dom: interfețe – IDL

Oferă suport pentru moștenire multiplă
prin intermediul interfețelor

specifică module, interfețe, metode, tipuri,
attribute, excepții, constante

www.w3.org/TR/WebIDL-1/ (*W3C Recommendation*, 2016)

heycam.github.io/webidl/ (în lucru, 26 martie 2019)



dom: **interfețe** – exemplu

Specificarea interfeței **NodeList**

```
interface NodeList {  
    getter Node? item (unsigned long index);  
    readonly attribute unsigned long length;  
};
```

dom: interfețe – exemplu

Specificarea interfeței **NodeList**

```
interface NodeList {  
    getter Node? item (unsigned long index);  
    readonly attribute unsigned long length;  
};
```

metodă cu un parametru;
rezultat: o valoare de tip **Node**

proprietate *read-only*
de tip întreg lung fără semn

dom: interfețe – exemplu

Specificarea interfeței **Attr**

```
interface Attr : Node {  
    readonly attribute DOMString name;  
    readonly attribute boolean specified;  
    attribute DOMString value;  
};
```

dom: **interfețe** – exemplu

Specificarea interfeței **Attr**

Attr extinde Node



```
interface Attr : Node {  
  readonly attribute DOMString name;  
  readonly attribute boolean specified;  
  attribute DOMString value;  
};
```

3 proprietăți

dom: core

Un document \equiv ierarhie de **obiecte-nod**
care pot implementa interfețe (specializate)

dom: core

Un document \equiv ierarhie de **obiecte-nod**
care pot implementa interfețe (specializate)

nodurile posedă descendenți ori sunt noduri frunză

parcurgerea arborelui se realizează
în preordine, în adâncime (*depth-first*)

a se (re)vedea
XML Infoset

dom: core

Accesul la date

- liste de noduri, attribute, valori,... –
- se realizează recurgându-se la metodele specifice
fiecărui tip de noduri ale arborelui

dom: core – tipuri de noduri

Document	Element, ProcessingInstruction, Comment, DocumentType
Document Fragment	Element, ProcessingInstruction, Comment, Text, CDATASection,...
Element	Element, Text, Comment, CDATASection, EntityReference,...
Attr	Text, EntityReference
Text	– (nod frunză al arborelui DOM)

dom: core

Interfețe fundamentale:

DOMException

gestionează setul de excepții de procesare

dom: core

Interfețe fundamentale:

DOMImplementation

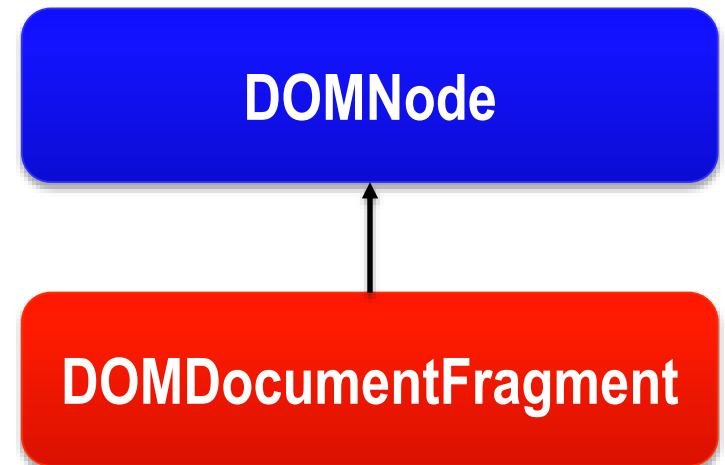
furnizează detalii despre implementarea curentă

dom: core

Interfețe fundamentale:

DocumentFragment : **Node**

acces la fragmente de arbore
(obiect minimal fără nod rădăcină)



dom: core

Interfețe fundamentale:

Document

oferă acces la document

pentru consultare și/sau modificare

dom: core

Interfețe fundamentale:

Document

proprietăți

doctype, implementation, documentElement

dom: core

Interfețe fundamentale:

Document

proprietăți

doctype, implementation, documentElement



acces la
elementul-rădăcină

dom: core

Interfețe fundamentale:

Document

metode createElement(), createTextNode(),
createAttribute(), getElementsByTagName(),
getElementsByTagNameNS(), getElementById(),
createElementNS(), importNode() etc.

dom: core

Interfețe fundamentale:

Node

acces la nodurile arborelui

tipuri de noduri – *e.g.*, Document, Element, CharacterData (Text, Comment, CDATASection), DocumentFragment, DocumentType, EntityReference,... – prelucrate în mod similar

dom: core

Interfețe fundamentale:

Node

proprietăți: nodeName nodeValue.nodeType
parentNode parentElement
childNodes firstChild lastChild
previousSibling nextSibling

Tipuri de noduri (proprietatea nodeType)	Valoare
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

dom: core

Interfețe fundamentale:

Node

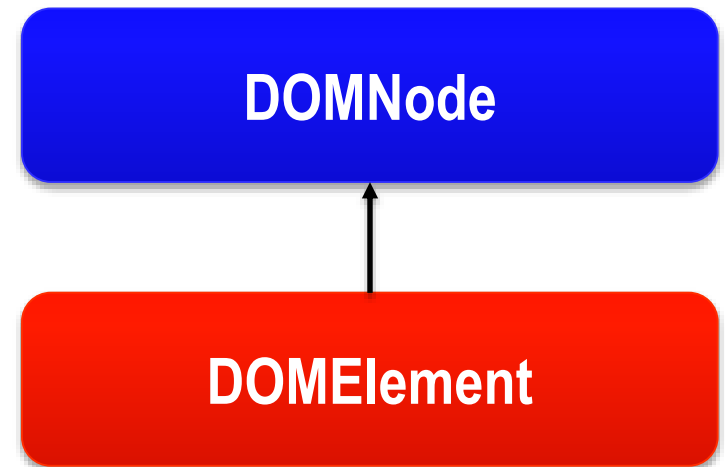
metode `insertBefore()`, `appendChild()`,
`replaceChild()`, `removeChild()`, `cloneNode()`,
`hasChildNodes()`, `isEqualNode()`, `isSameNode()`

dom: core

Interfețe fundamentale:

Element

facilitează accesul la elementele XML



dom: core

Interfețe fundamentale:

Element

proprietăți: **tagName** **id**

childNodes **firstChild** **lastChild**

attributes

nextElementSibling

previousElementSibling

dom: core

Interfețe fundamentale:

Element

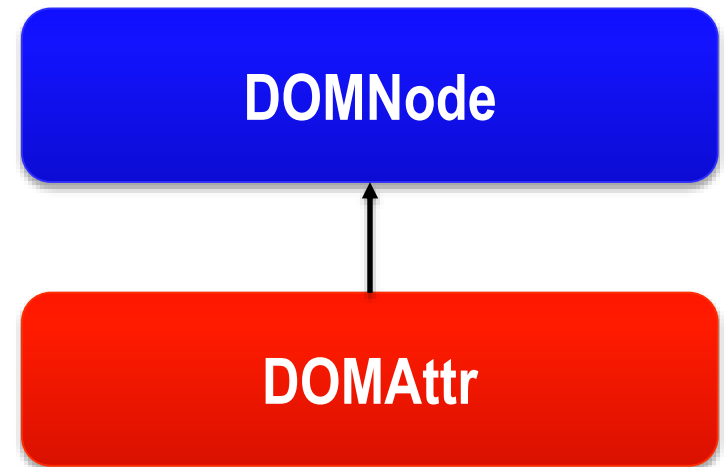
metode `getAttribute()`, `getAttributeNS()`,
`getAttributeNames()`, `getAttributeNode()`,
`setAttribute()`, `setAttributeNS()`,
`removeAttribute()`, `removeAttributeNS()`,
`hasAttribute()`, `hasAttributeNS()`,...

dom: core

Interfețe fundamentale:

Attr : Node

acces la atributele
unui element



dom: core

Interfețe fundamentale:

NodeList – acces la colecții de noduri via indecși

NamedNodeMap – acces pe baza cheilor

(în cazul HTML, pentru liste de attribute)

proprietate: **length**

metode: **item()** **getNamedItem()** **getNamedItemNS()**
setNamedItem() **removeNamedItem()** etc.

dom: html

DOM HTML extinde DOM Core

specializarea interfețelor și oferirea de suport obiectual
pentru prelucrarea documentelor HTML

standardizează procesarea paginilor Web
(*e.g.*, în cadrul navigatorului)

Markup to test ([permalink](#), [save](#), [upload](#), [download](#), [hide](#)):

```
<!DOCTYPE html>
<html><head><title>DOM</title></head><body><article><h1 class="h">Web</h1>
Document <em>Object</em> Model</article></body>
```

[DOM view](#) ([hide](#), [refresh](#)):

```
└ DOCTYPE: html
  └ HTML
    └ HEAD
      └ TITLE
        └ #text: DOM
    └ BODY
      └ ARTICLE
        └ H1 class="h"
          └ #text: Web
        └ #text:
          └ IMG src="image" alt="Web"
        └ #text: Document
        └ EM
          └ #text: Object
        └ #text: Model
```

în cazul HTML, numele elementelor
sunt disponibile cu litere mari (*capitals*)

un document HTML și arborele DOM corespunzător
reprezentat via **Live DOM Viewer**
software.hixie.ch/utilities/js/live-dom-viewer/

Markup to test ([permalink](#), [save](#), [upload](#), [download](#), [hide](#)):

```
<!DOCTYPE html>
<html><head><title>DOM</title></head><body><article><h1 class="h">Web</h1>
Document <em>Object</em> Model</article></body>
```

[DOM view](#) ([hide](#), [refresh](#)):

```
DOCTYPE: html
HTML
├── HEAD
│   └── TITLE
│       └── #text: DOM
├── BODY
│   └── ARTICLE
│       ├── H1 class="h"
│       │   └── #text: Web
│       ├── #text:
│       ├── IMG src="image" alt="Web"
│       ├── #text: Document
│       ├── EM
│       │   └── #text: Object
│       └── #text: Model
```

De ce apare
și acest nod?

un document HTML și arborele DOM corespunzător
reprezentat via **Live DOM Viewer**
software.hixie.ch/utilities/js/live-dom-viewer/

Arborele DOM asociat documentului HTML
poate fi accesat/alterat via obiectul **document**

instanță a clasei implementând interfața **HTMLDocument**

```
interface HTMLDocument : Document {
    attribute DOMString title;           // titlul documentului
    readonly attribute DOMString referrer; // adresa resursei ce referă pagina
    readonly attribute DOMString domain; // domeniul de care aparține
    readonly attribute DOMString URL;    // URL-ul absolut al documentului
    attribute HTMLElement body;         // acces la elementul <body>
    readonly attribute HTMLCollection images; // lista tuturor imaginilor
    readonly attribute HTMLCollection links;  // lista tuturor legăturilor
    readonly attribute HTMLCollection forms; // lista tuturor formularelor

    attribute DOMString cookie;           // acces la cookie-uri
    // emite o excepție dacă e asignată o valoare

    void open (); // deschide un flux de scriere (alterează DOM-ul curent)
    void close (); // închide fluxul de scriere și forțează redarea conținutului
    void write (in DOMString text); // scrie un șir de caract. (e.g., cod HTML)
    void writeln (in DOMString text); // idem, dar inserează și new line
    NodeList getElementsByName (in DOMString numeElement);
    // furnizează o listă de elemente conform unui nume de tag
};
```


dom: html

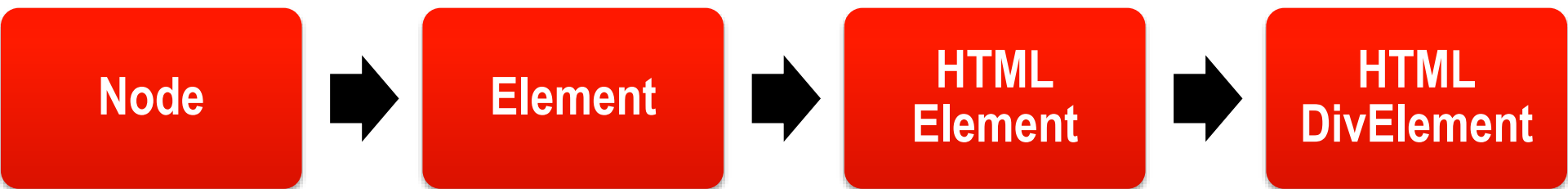
Interfața **HTMLCollection** reprezintă o listă de noduri HTML

un nod poate fi accesat folosind un index numeric
sau pe baza unui identificator – stabilit via atributul **id**

```
interface HTMLCollection {  
  readonly attribute unsigned long length;    // oferă numărul nodurilor din listă  
  Node      item (in unsigned long index);    // oferă un nod via un index numeric  
  Node      namedItem (in DOMString name);    // furnizează un nod pe baza numelui  
};
```

Interfața **HTMLElement** o extinde
pe cea generală oferită de DOM – nivelul 2

- fiecare element HTML derivă din ea
- ▶ o interfață specifică fiecărui element HTML



```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Web</p>
```

```
<div>
```

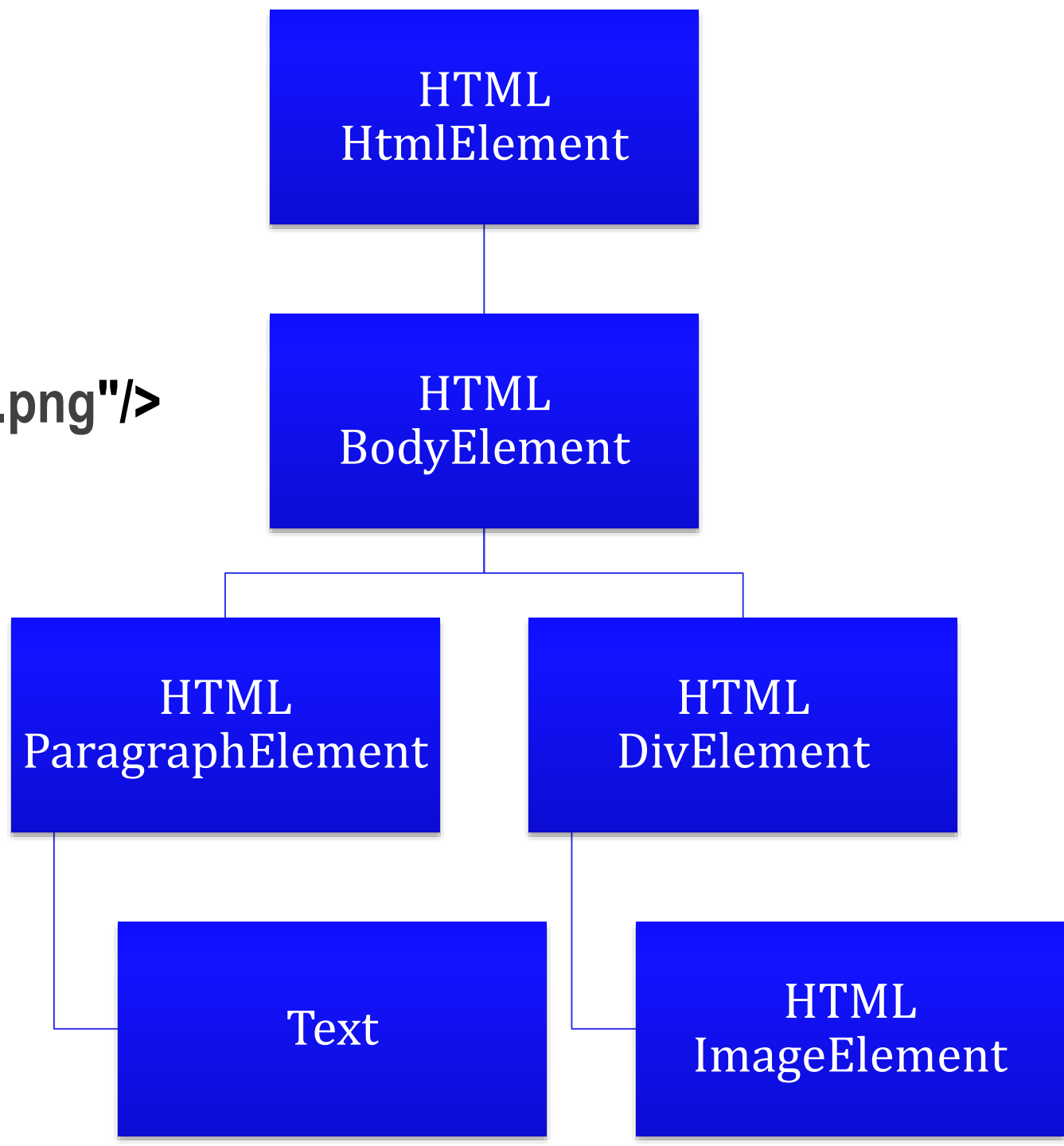
```

```

```
</div>
```

```
</body>
```

```
</html>
```



<!DOCTYPE html>

<html>

<body>

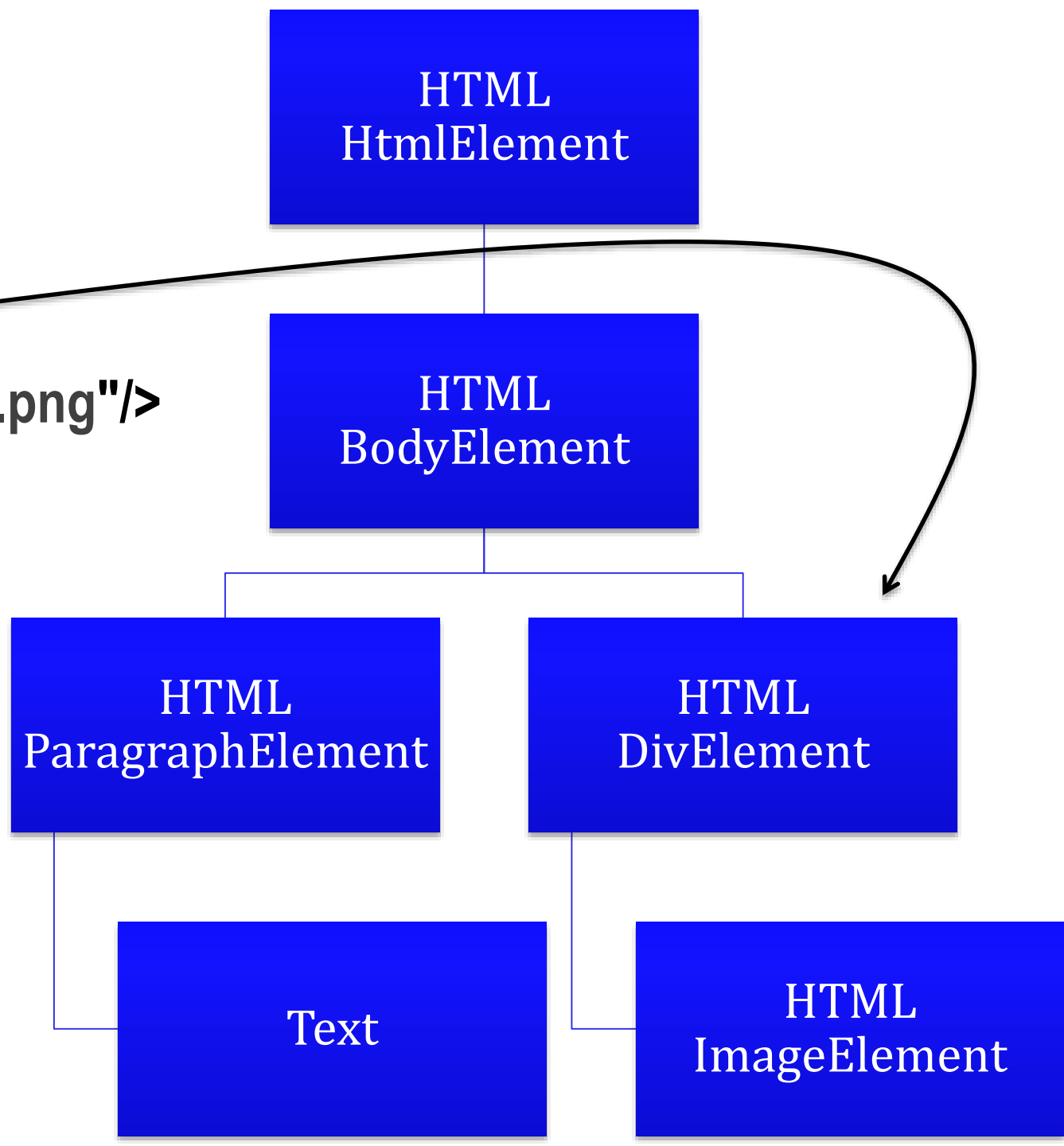
<p>Web</p>

<div>

</div>

</body>

</html>



```

// un element HTML generic
interface HTMLElement : Element {
    attribute DOMString    id;                // identificator asociat elementului
    attribute DOMString    title;             // titlu explicativ
    attribute DOMString    lang;             // limba în care e redactat conținutul
    attribute DOMString    className;        // numele clasei CSS folosite pentru redare
};

// specifică un formular Web
interface HTMLFormElement : HTMLElement {
    readonly attribute HTMLCollection elements; // elementele HTML incluse în formular
    readonly attribute long length; // numărul câmpurilor formularului
    attribute DOMString action; // URI-ul resursei ce procesează datele
    attribute DOMString enctype; // tipul MIME de codificare a datelor
                                     // (application/x-www-form-urlencoded)
    attribute DOMString method; // metoda HTTP folosită (GET sau POST)
    void submit(); // trimite date URI-ului definit de 'action'
};

// interfața DOM pentru elementul <img/> (e.g., conținut grafic raster: GIF, JPEG, PNG)
interface HTMLImageElement : HTMLElement {
    attribute DOMString alt; // text alternativ descriind conținutul grafic
    attribute DOMString src; // URL-ul resursei reprezentând imaginea
};

```

dom: html

Aspecte specifice:

innerHTML

proprietate – mutabilă – ce furnizează marcasele HTML
din cadrul unui nod de tip **Element**

utilizare

nerecomandabilă

dom: html

Aspecte specifice:

textContent

proprietate ce furnizează/stabilește conținutul textual
al nodului și posibililor descendenți

dom: nivelul 2

Extinde funcționalitățile DOM1

crearea unui obiect **Document**
copierea unui nod dintr-un document în altul
și multe altele...



specificația
DOM 2 Core

dom: nivelul 2

Extinde funcționalitățile DOM1

alte facilități:

controlul aplicării foilor de stiluri CSS

tratarea evenimentelor

specificarea filtrelor și iteratorilor
(parcurgeri sofisticate de arbori DOM)

dom: nivelul 2

Suport pentru procesarea foilor de stiluri CSS

StyleSheet

StyleSheetList

MediaList

DocumentStyle

detalii la www.w3.org/TR/DOM-Level-2-Style

dom: nivelul 2

Suport pentru procesarea foilor de stiluri CSS

pentru HTML5, se realizează pe baza unui model
obiectual specific: **CSSOM** (*CSS Object Model*)

specificație în lucru (*draft*) – 21 ianuarie 2019

drafts.csswg.org/cssom/

dom: nivelul 2

Suport pentru procesarea foilor de stiluri CSS

actualmente, modificarea proprietăților de stil
se poate realiza via proprietatea **HTMLElement.style**

// asocierea mai multor stiluri CSS

```
elem.style.cssText = "color: blue; border: 1px solid #000";
```

// similar cu:

```
elem.setAttribute("style", "color: blue; border: 1px solid #000;");
```

dom: nivelul 2

Tratarea evenimentelor

definirea de activități (*callback-uri*) executate
la apariția unui eveniment

eveniment = acțiune produsă în cadrul mediului de
execuție în urma căreia programul va putea reacționa

dom: nivelul 2

Tratarea evenimentelor

definirea de activități (*callback-uri*) executate la apariția unui eveniment

eveniment = acțiune produsă în cadrul mediului de execuție în urma căreia programul va putea reacționa

în cazul *browser*-ului Web, codul JavaScript invocat la apariția unui eveniment va putea fi încapsulat într-o funcție de tratare a acestuia (*event handler*)

dom: nivelul 2

Tratarea evenimentelor

descrierea arborescentă a fluxului de evenimente

capture versus bubble

dom: nivelul 2

Tratarea evenimentelor

descrierea arborescentă a fluxului de evenimente

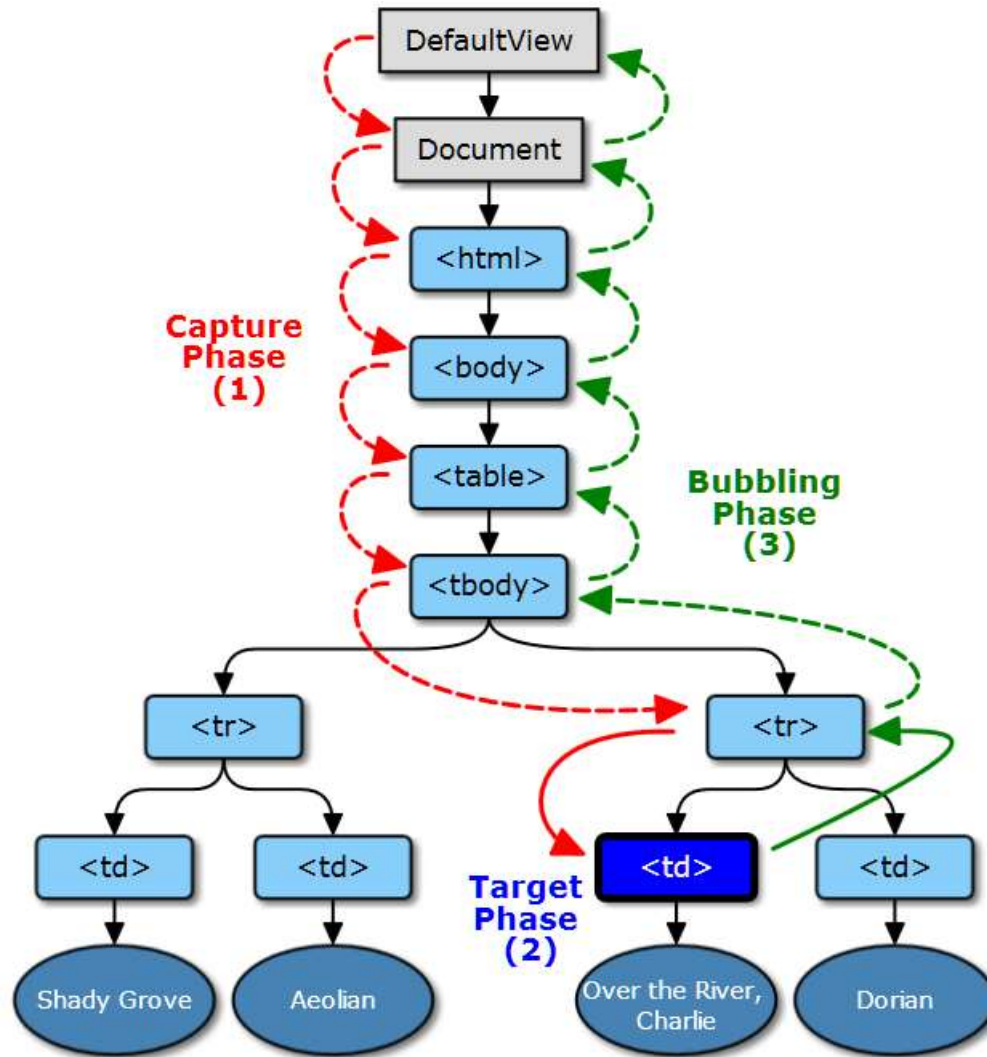
tratarea evenimentului se poate face pornind
de la rădăcină până la obiectul-țintă – *capture phase*

dom: nivelul 2

Tratarea evenimentelor

descrierea arborescentă a fluxului de evenimente

tratarea evenimentului poate avea loc atunci când evenimentul e propagat de la obiectul unde a survenit până la entitățile superioare lui – *bubbling phase*



fluxul de evenimente (T. Leithead *et al.*, 2012)

a se studia și W. Page, *An Introduction to DOM Events* (2013)
www.smashingmagazine.com/2013/11/an-introduction-to-dom-events/

dom: nivelul 2

Tratarea evenimentelor

se va utiliza un set standard de evenimente

www.w3.org/TR/DOM-Level-2-Events

dom: nivelul 2

Tipuri de evenimente:

de interfață – context: interacțiunea cu utilizatorul

mouse: **click, mousedown, mouseup, mouseover, mousemove**

tastatură: **keypress, keydown, keyup**

uzual, folosite în
contextul HTML

dom: nivelul 2

Tipuri de evenimente:

referitoare la interacțiunea cu *browser*-ul
specifice HTML (document Web ori formular)

**load, unload, abort, error,
select, submit, focus, blur, resize, scroll**

dom: nivelul 2

Tratarea evenimentelor

sunt puse la dispoziție interfețele:

EventTarget

EventListener

Event – minimal: UIEvent și MouseEvent

dom: nivelul 2

Traversarea arborilor DOM

se specifică interfețele opționale

TreeWalker

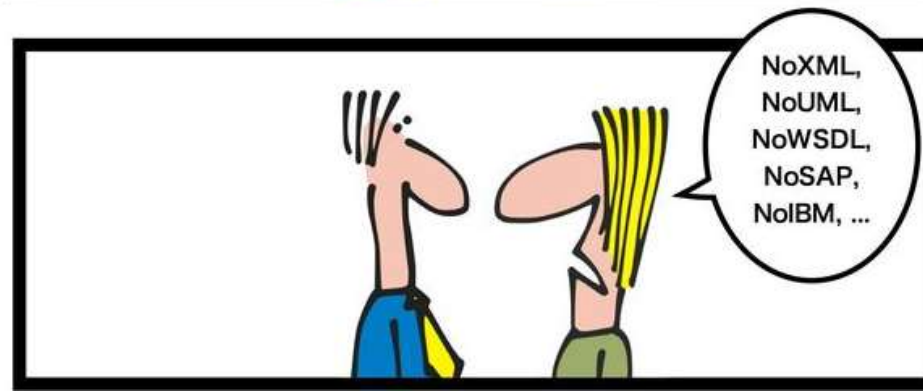
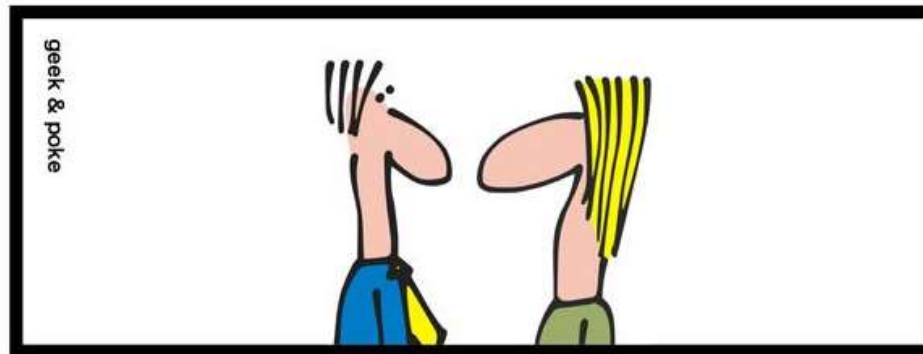
NodeIterator

Filter

www.w3.org/TR/DOM-Level-2-Traversal-Range

(în loc de) pauză

RECENTLY DURING THE
JOB INTERVIEW



dom: nivelul 3

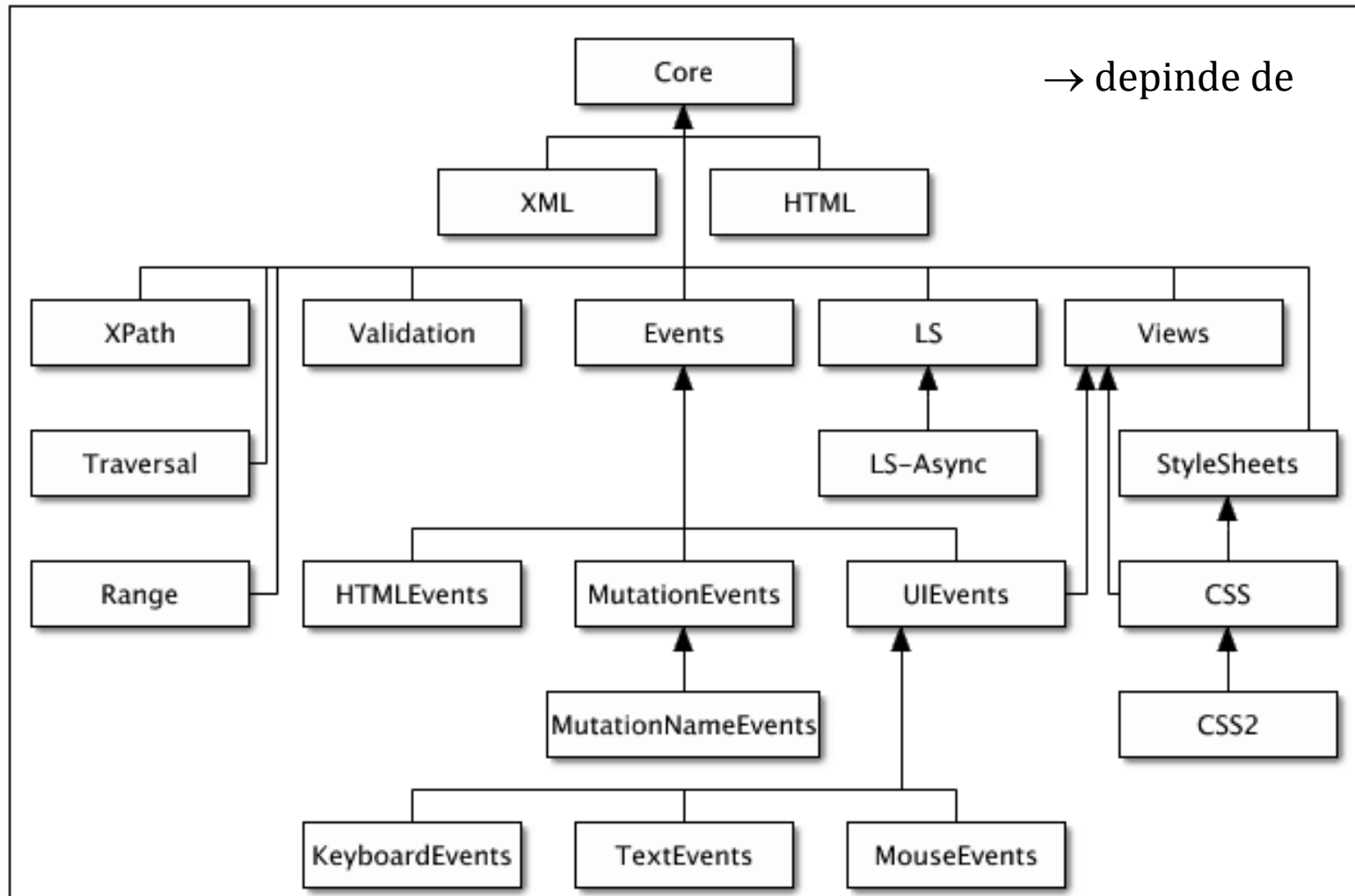
Extinde DOM 2, oferindu-se interfețe
pentru manipularea XML via module DOM

un modul pune la dispoziție o facilitate particulară

dom: nivelul 3

Modulul **Core** include interfețele fundamentale ce trebuie implementate de toate implementările DOM conformându-se standardului

www.w3.org/TR/DOM-Level-3-Core



module disponibile: XML, HTML, XPath, Traversal, Range, Validation, Events, Views, Load & Save, Stylesheet,...

dom: nivelul 3

Module DOM 3 standardizate
DOM Load & Save

interfețe puse la dispoziție:

LSParser, LSInput, LSSerializer, LSOutput

www.w3.org/TR/DOM-Level-3-LS

dom: nivelul 3

Module DOM 3 standardizate
DOM Validation

oferă funcționalități de creare/editare (automată)
de documente conformându-se
unor scheme de validare

www.w3.org/TR/DOM-Level-3-Val

dom: nivelul 4

Unifică DOM3 Core, Element Traversal,
Selectors API – nivelul 2, DOM3 Events

noi interfețe: **ParentNode**, **ChildNode**, **Elements**,...

suport și pentru specificarea de evenimente proprii
via interfața **CustomEvent**

dom: nivelul 4

Interfața **Element** include noile metode:
getElementsByClassName () și **matches ()**

dom: nivelul 4

Selectors API

acces la diverse date via selectorii CSS cu metodele
`query()` `queryAll()` `querySelector()` `querySelectorAll()`

```
// toate elementele <li> selectate via CSS (date de tip NodeList)
var elemente = document.querySelectorAll ("ul.menu > li");
for (var i = 0; i < elemente.length; i++) {
    prelucrează (elemente.item (i));           // procesăm fiecare nod
}
```

www.w3.org/TR/selectors-api/

- Introduction & Administrative

h2 704 × 45.9

- Lecture: Web Application Development: Concepts & Vision

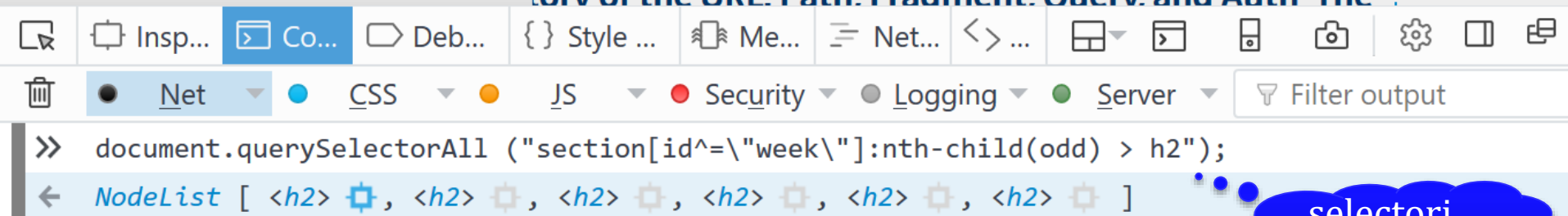
remember: Web Technologies [RO]

Web, architecture, social Web, data, modeling, standards

- Lab: Server-side Web development – Web Developer?!, Code in the Clouds, What Technical Details Should a Programmer of a Web Application Consider?, Awesome Lists

- Additional resources: Tim Berners-Lee: The Next Web, 25 Years of Web, Ideas That Changed the Web, The History of the URL: Domain, Protocol, Path, Fragment, Query and Auth The

chrome://devtools/content/webconsole/webconsole.xul#



Click to select the node in the inspector

selector
CSS3

exemplificare – folosim consola *browser*-ului Web:

```
document.querySelector ("section[id^=\"week\"] :nth-child(odd) > h2");
```

dom: implementări

DOMDocument – clasă PHP
php.net/manual/en/book.dom.php

HXT (*Haskell XML Toolbox*) – procesări DOM în Haskell
wiki.haskell.org/HXT

JAXP (*Java Architecture for XML Processing*)
parte integrantă din J2SE (`javax.xml.*`)
docs.oracle.com/javase/tutorial/jaxp/

jsdom – modul Node.js: github.com/jsdom/jsdom

JSXML – bibliotecă JavaScript: jsxml.net

dom: implementări

QDOM – la Qt (C++): doc.qt.io/qt-5/qdomdocument.html

libxml – procesor XML scris în C: xmlsoft.org

baza unor biblioteci pentru C++, Perl, PHP, Python, Ruby,...

MSDOM – procesări XML în C++ (parte din MSXML SDK)
[msdn.microsoft.com/en-us/library/ms763742\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms763742(v=vs.85).aspx)

org.w3c.dom.Document – interfață DOM (Android)
developer.android.com/reference/org/w3c/dom/Document.html

script::dom – modul Rust: doc.servo.org/script/dom/

TinyXML2 – bibliotecă de procesare DOM pentru C++
github.com/leethomason/tinyxml2

dom: implementări

Xerces DOM API – platformă de procesare XML
(C++ și Java): xerces.apache.org

XmlDocument – clasă .NET Framework (C# *et al.*)
docs.microsoft.com/en-us/dotnet/api/system.xml.xmldocument

XMLDocument – clasă Objective-C / Swift
developer.apple.com/documentation/foundation/xmldocument

XML::DOM – modul Perl: search.cpan.org/perldoc?XML::DOM
bazat pe procesorul Expat: libexpat.github.io

xml.dom – modul Python: docs.python.org/3/library/xml.dom.html

dom: api-uri particulare (exemple)

node-xmpp – biblioteca Node.js (JavaScript) pentru XMPP
xmppjs.org

OpenDocument Format (ODF) SDKs – procesarea documentelor ODF în C#, Java, PHP, Python,...
opendocumentformat.org/developers/

Open Street Map API – acces la datele XML oferite de serviciul cartografic liber OpenStreetMap
wiki.openstreetmap.org/wiki/OSM_XML
wiki.openstreetmap.org/wiki/API

dom: api-uri particulare (exemple)

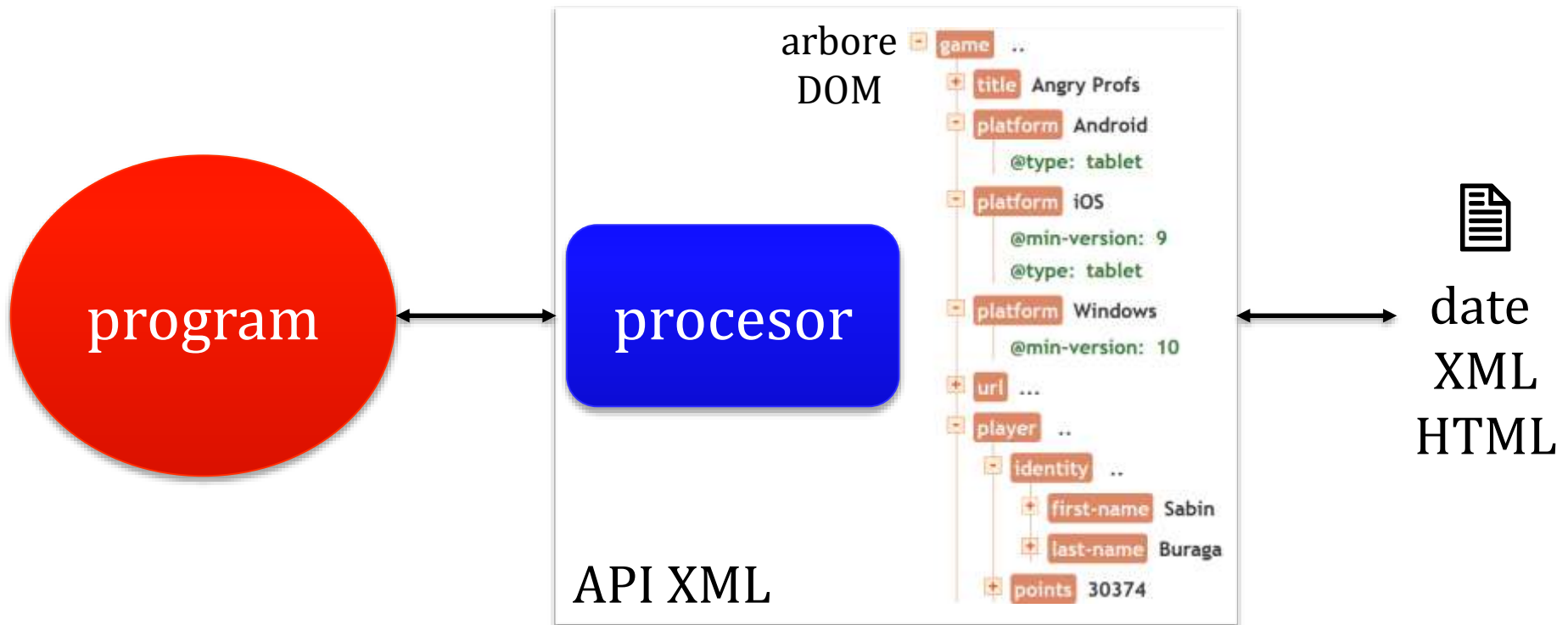
SVG DOM API – procesarea documentelor SVG în Java
(Apache Batik): xmlgraphics.apache.org/batik/
developer.mozilla.org/Web/API/Document_Object_Model#SVG_interfaces

Twiml (*Twilio Markup Language*) API – accesarea
serviciilor de telefonie Twilio via biblioteci C#, Java,
Node.js, PHP, Python, Ruby: www.twilio.com/docs/voice/twiml

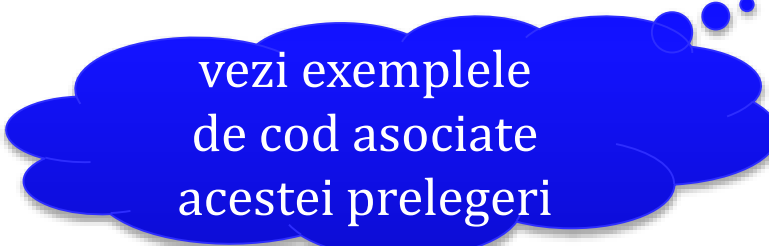
Xamarin.Forms – crearea via C# de interfețe-utilizator
specificate în XAML pentru Android, iOS și Windows
www.xamarin.com/forms

dom: implementări

Procesarea documentelor XML/HTML



Studiu de caz: preluarea informațiilor referitoare la lista proiectelor propuse



vezi exemplele
de cod asociate
acestei prelegeri


```
try {  
    $doc = new DomDocument; // instanțiem un obiect DOM  
    $doc->load ("projects.xml"); // încărcăm documentul XML  
    // afișăm informații privitoare la proiecte: titlul + clasa (dacă există)  
    $projs = $doc->getElementsByTagName("project");  
    foreach ($projs as $proj) { // preluăm nodurile-element <title>  
        $titles = $proj->getElementsByTagName("title");  
        foreach ($titles as $title) {  
            echo "Proiect: " . $title->nodeValue;  
        }  
        // verificăm dacă există specificată clasa proiectului  
        if ($proj->hasAttribute("class")) {  
            echo " de clasa " . $proj->getAttribute("class");  
        }  
    }  
} catch (Exception $e) {  
    die ("Din păcate, a survenit o excepție.");  
}
```

procesări DOM
în limbajul PHP

```
import urllib
import xml.dom

from xml.dom.minidom import parse

try:
    doc = urllib.urlopen('projects.xml') # încărcăm documentul
    dom = parse(doc)                     # îl procesăm...
    # parcurgem elementele <project> și oferim informații despre fiecare
    for proj in dom.getElementsByTagName('project'):
        print 'Proiectul ' + proj.childNodes[1].firstChild.nodeValue + \
            ' este de clasă: ' + proj.getAttribute('class')
except Exception:
    print 'Din păcate, a survenit o excepție.'
```

procesări DOM în limbajul Python

```

using System.Xml;
...
try {
    doc = new XmlDocument(); // instanțiem un document XML
    doc.Load("projects.xml"); // pentru a fi încărcat
    // afișăm informații privitoare la proiecte: titlu și clasă
    XmlNodeList projs = doc.GetElementsByTagName("project");
    foreach (XmlElement proj in projs) {
        // selectăm nodurile <title> via o expresie XPath
        XmlNodeList titles = proj.SelectNodes("./title"); // DOM nivelul 2
        foreach (XmlElement title in titles) {
            Console.WriteLine("Proiect: {0} ", title.InnerXml);
        }
        if (proj.HasAttribute("class") == true) { // există clasa specificată?
            Console.WriteLine("de clasa '{0}'.", proj.GetAttribute("class"));
        }
    }
} catch (Exception e) {
    // a survenit o excepție...
}

```

procesări DOM
în C# (.NET)

```

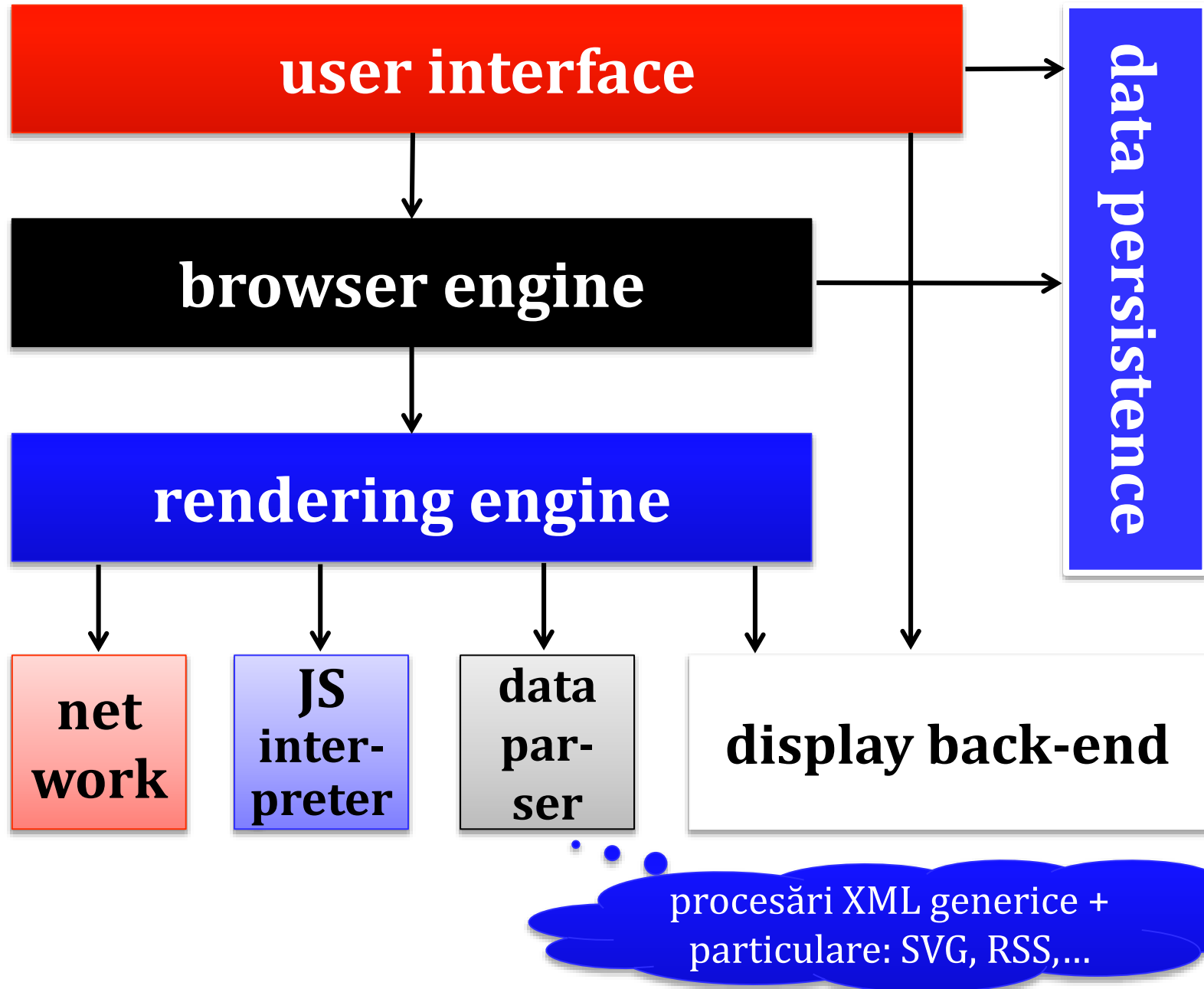
import org.w3c.dom.*; import javax.xml.parsers.*;
...
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder docb = dbf.newDocumentBuilder();
Document doc = docb.parse("projects.xml");
// traversăm recursiv arborele DOM
traversează (doc.getDocumentElement());
...
static private void traversează (Node nod) {
    // afișăm numele nodurilor de tip element, plus numărul de attribute
    if (nod.getNodeType() == Node.ELEMENT_NODE) {
        System.out.println ("Elementul " + nod.getNodeName() + " are " +
            nod.getAttributes().getLength() + " attribute.");
    }
    Node copil = nod.getFirstChild();
    if (copil != null) { traversează (copil); }
    copil = nod.getNextSibling();
    if (copil != null) { traversează (copil); }
}

```

procesări DOM
în Java

Prelucrarea documentelor XML prin DOM la nivel de client Web?

Prelucrarea documentelor XML în *browser*-ul Web



dom: browser

Accesarea/procesarea documentelor HTML și XML
– fără validare – se realizează via DOM de programe

JavaScript (ECMAScript) interpretate de navigatorul Web

pentru detalii, a se studia prelegerile materiei

Dezvoltarea aplicațiilor Web la nivel de client

profs.info.uaic.ro/~busaco/teach/courses/cliw/web-film.html

dom: browser

Exemplul #1:
crearea dinamică de marcaje HTML
via un program JavaScript


```
// funcție care generează un număr de elemente HTML
// pe care le adaugă elementului identificat prin 'identificator'
function genereazaElemente(numarElem, numeElem, identificator) {
    for (var it = 0; it < numarElem; it++) {
        // creăm un element specific
        var element = document.createElement(numeElem);
        // ...și-i atașăm un nod text
        var text = document.createTextNode("Salut, lumea...");
        element.appendChild(text);
        // adăugăm nodul creat
        document.getElementById(identificator).appendChild(element);
    }
}

genereazaElemente(3, "div", "continut"); // 3 <div>-uri
genereazaElemente(2, "p", "lumi");      // 2 paragrafe (<p>)
```

```
<div id="lumi"></div>
<h1 id="continut"></h1>
```

a se studia exemplele
din arhivă



Salut, lumea #0...

Salut, lumea #1...

Salut, lumea #1...

Salut, lumea #2...

Salut, lumea #0...

Inspector Console Debugger Network

Search HTML

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <!--acest element va include cod generat dinamic-->
    <div id="lumi">
    </div>
    <h1 id="continut">
      <div>Salut, lumea #1...</div>
      <div>Salut, lumea #2...</div>
      <div>Salut, lumea #0...</div>
    </h1>
    <!--
      apelam functia JavaScript
      alterarea DOM-ului asociat
    -->
    <script type="application/javascript">
    </script>
  </body>
</html>
```

html > body > h1#continut > div

Filter output

```
firstChild: #text "Salut, lumea #2..."
firstElementChild: null
hidden: false
id: ""
innerHTML: "Salut, lumea #2..."
innerText: "Salut, lumea #2..."
isConnected: true
isContentEditable: false
lang: ""
lastChild: #text "Salut, lumea #2..."
lastElementChild: null
localName: "div"
```

arborele DOM corespunzător
codului HTML
generat prin program

inspectarea valorilor
proprietăților DOM

dom: browser

Exemplul #2:
tratarea unor evenimente
generate de interacțiunea cu utilizatorul

```
// Adaptare după https://eloquentjavascript.net/14_event.html
const trateazaEveniment = ev => {
  // plasăm un 'punct' la coordonatele cursorului mouse-ului
  let pct = document.createElement ('div');
  pct.className = (ev.type === 'dblclick') ? 'punct roz' : 'punct';
  pct.style.left = (ev.pageX - 5) + 'px';
  pct.style.top = (ev.pageY - 5) + 'px';
  document.body.appendChild (pct);
  console.log (` ${ev.type}: Am plasat un punct
                la coord. (${ev.pageX}, ${ev.pageY}).`);
};
// "ascultăm" evenimentele click și dblclick
document.addEventListener ('click', trateazaEveniment);
document.addEventListener ('dblclick', trateazaEveniment);
```

vezi exemplul
complet în arhivă

tratarea evenimentelor **click** și **dblclick**

Un click (dublu) de mouse, te rog...

div.punct | 10x10

Elements

Console

Sources

Network

Performance

Memory

Application

»

⋮

✕

<!DOCTYPE html>

<html>

▶<head>...</head>

...▼<body> == \$0

<p>Un click (dublu) de mouse, te rog...</p>

▶<script type="application/javascript">...

</script>

<div class="punct" style="left: 311px; top: 107px;"></div>

<div class="punct" style="left: 143px; top: 181px;"></div>

<div class="punct" style="left: 143px; top: 181px;"></div>

<div class="punct roz" style="left: 143px; top: 181px;"></div>

<div class="punct" style="left: 95px; top: 71px;"></div>

<div class="punct" style="left: 694px; top: 22px;"></div>

</body>

Styles

Computed

Event Listeners

»

🔄

☒ Ancestors

All

▼

☒ Framework listeners

▼ click

▼ document [clicks.html:25](#)

useCapture: false

passive: false

once: false

▼ handler: ev => {...}

▶ [[Scopes]]: Scopes[2]

[[FunctionLocation]]: <unknown>

▶ __proto__: f ()

name: "trateazaEveniment"

length: 1

caller: (...)

arguments: (...)

▶ dblclick

html

body

dom: browser

Exemplul #3:
parcurgerea arborelui DOM
corespunzător unui document HTML
pentru a selecta anumite elemente

```
// instanțiem un obiect TreeWalker pentru parcurgere
let calator = document.createTreeWalker(document.body,
  NodeFilter.SHOW_ELEMENT, // selectăm doar nodurile de tip element
  { acceptNode: nod => { // ...și le filtrăm (doar <p>, <div> și <strong>)
    if (nod.nodeName === 'P' || nod.nodeName === 'DIV' ||
      nod.nodeName === 'STRONG') return NodeFilter.FILTER_ACCEPT;
  }});

let noduri = [ ];

// baleiem toate nodurile găsite și le plasăm în tabloul 'noduri'
while(calator.nextNode()) noduri.push(calator.currentNode);
// listăm nodurile găsite
let elem = document.getElementById('info');
noduri.forEach(nod => {
  // plasăm informațiile în DOM, în <div> înainte de ultimul nod copil
  elem.insertAdjacentText('beforeend', nod.outerHTML + "\u25CF");
  // și la consola browser-ului Web
  console.log(`Element ${nod.nodeName}: ${nod.textContent}`);
});
```


HTML +

```

<html>
<head>
  <meta charset="utf-8" />
  <title>Parcurgerea arborelui DOM</title>
</head>
<body>
  <!-- -->
  <p><strong>JS</strong>
    + <strong>DOM</strong>
    = &#x1f49a;</p>
  <div id="info"></div>
</body>
</html>

```

JavaScript +

```

// instantiem un obiect TreeWalker pentru a parcurge arborele DOM
let calator = document.createTreeWalker(
  document.documentElement,
  // selectam doar elementele
  NodeFilter.SHOW_ELEMENT,
  // ...si le filtram (acceptam doar <p> si <div>)
  {
    acceptNode: nod => {
      if (nod.nodeName == 'P' ||
          nod.nodeName == 'DIV' ||
          nod.nodeName == 'STRONG')
        return NodeFilter.FILTER_ACCEPT;
    }
  }
);

let noduri = [];

// baleiem toate nodurile gasite si le plasam in tabloul 'noduri'
while (calator.nextNode())
  noduri.push(calator.currentNode);

// listam nodurile gasite
let elem = document.getElementById('info');
noduri.forEach(nod => {
  // plasam informatiile in DOM, in <div>
  // inainte de ultimul nod copil
  elem.insertAdjacentText('beforeend', nod.outerHTML + "\u25CF");
  // si la consola browser-ului Web
  console.log('Element ${nod.nodeName}: ${nod.textContent}');
});

```

Console

Run Clear

```

"Element P: JS
  + DOM
  = ❤️"

"Element STRONG: JS"

"Element STRONG: DOM"

"Element DIV: <p><strong>JS</strong>
  + <strong>DOM</strong>
  = ❤️"

</p>●<strong>JS</strong>●<strong>DOM
id="\info\ "&lt;p&gt;&lt;strong&gt;
JS&lt;/strong&gt;
+
&lt;strong&gt;DOM&lt;/strong&gt;
=
❤️&lt;/p&gt;●&lt;strong&gt;JS&lt;
/strong&gt;●&lt;strong&gt;DOM&lt;
/strong&gt;●</div>●"

```

Bin info

JS + DOM = ❤️

<p>JS + DOM = ❤️
 </p>●JS●DOM●<div id="info"><p>JS + DOM =
 ❤️</p>●JS●DOM< /strong>●</div>●

program disponibil la JSFiddle: jsfiddle.net/busaco/ofm958vr/

dom: browser

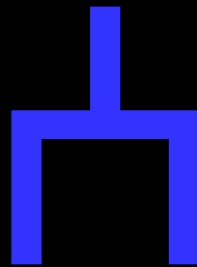
Se oferă suport și pentru transfer asincron de date între client (*browser*) și server Web

AJAX – Asynchronous JavaScript And XML
via obiectul **XMLHttpRequest** și/sau **Fetch API**



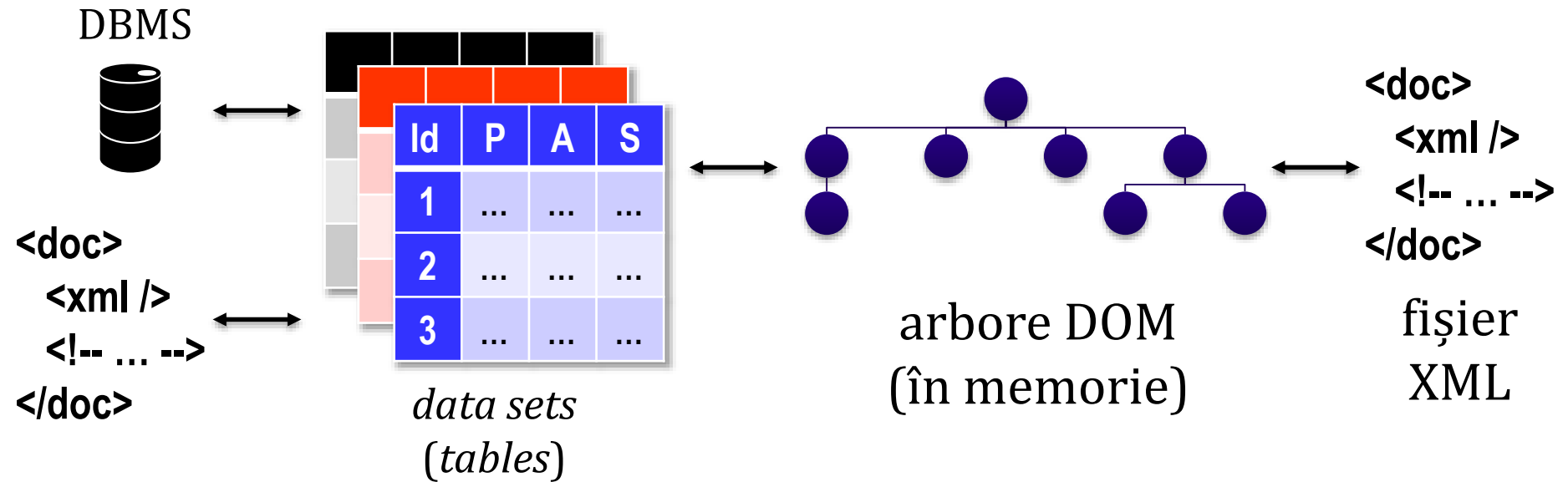
vezi cursurile
viitoare

rezumat



modelul DOM:

caracterizare, niveluri de specificare, exemple



episodul viitor:

procesări XML via SAX + prelucrări simplificate