

Императивна обработка на XML съдържание чрез Document Object Model (DOM)

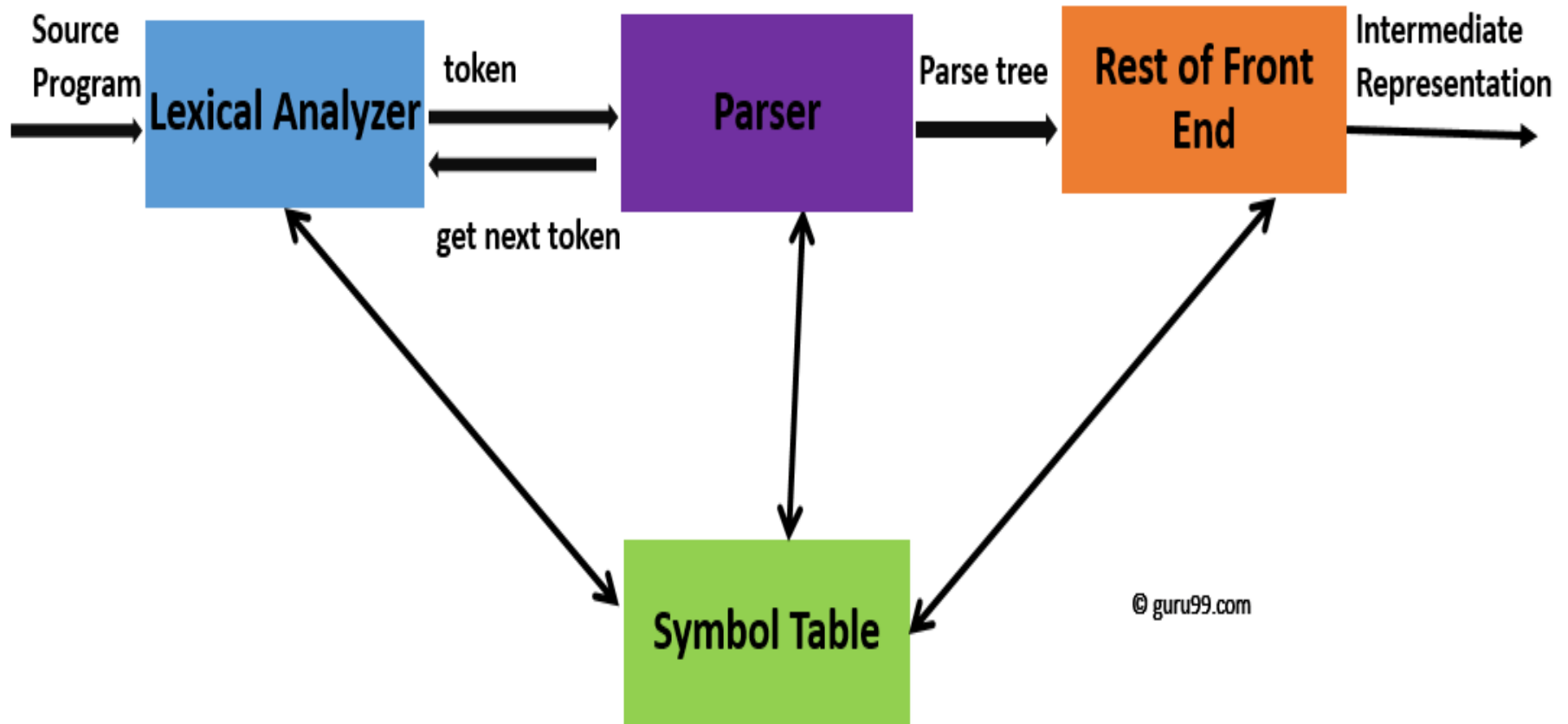
Предимства на DOM
DOM интерфейси
Използване
Пример
DOM, XSL и SAX



Анализ (разбор,
парсване) на текст



Процес на анализ (разбор, парсване) на текст



Използване на XML Parser

- Цел и приложение
- Три основни стъпки в използването на XML парсер:
 - Създаване на парсер-обект
 - Предаване на XML документа на парсера
 - Обработка на резултатите
- Принципно, генерацията на XML или друг формат е извън обхвата на парсерите

Типове XML парсъри

- Съществуват различни групи от парсери:
 - Валидиращи спрямо невалидиращи XML парсери
 - XML парсери, написани на конкретен език (Java, C++, Perl, etc.) без използване на определен API – напр. с използване на регулярни изрази
 - ОО-парсери, използващи Document Object Model (DOM) API
 - Управлявани по събития парсери, използващи Simple API for XML (SAX)
 - Базирани на итератори парсери, използващи Streaming API for XML (StAX)

Невалидиращи парсери

- Скорост и ефикасност

- Валидиращ XML парсер - с обработка на DTD/XML схема и последваща проверка дали всеки елемент в XML документа следва правилата на DTD или XML схема, изисква значително време и ресурси
- Ако целта е единствено да се намерят маркери (етикети) на елементи и да се извлече информация - използвайте невалидиращи парсери

Разбор (парсване) на XML съдържание

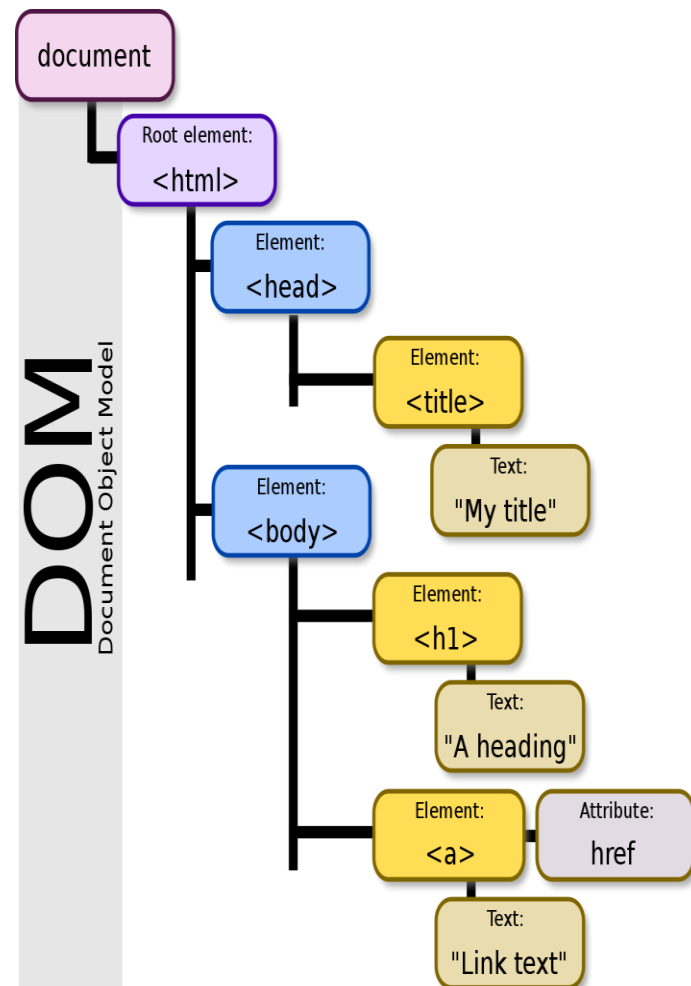
- Три широко-известни API's:
 - DOM (Document Object Model)
 - Дефинира логическо дърво, представящо анализирания XML документ
 - SAX (Simple API for XML)
 - Дефинира манипулатори (handlers), съдържащи методи за разбор на XML документа (*a la push*)
 - Streaming API for XML (StAX)
 - Парсване на XML, базирано на итератори (*a la pull*)
- Приложения без сложна манипулация на XML, но с ограничения по памет, могат да ползват SAX и StAX
- Структурната манипулация на XML елементи изисква използването на DOM

DOM - Document Object Model

- Комплект от приложни интерфейси за четене на XML файл в паметта и съхраняването му като дървовидна структура
- Абстрактните API дават възможност за изграждане, достъп и манипулиране на структурата и съдържанието на XML и HTML документи

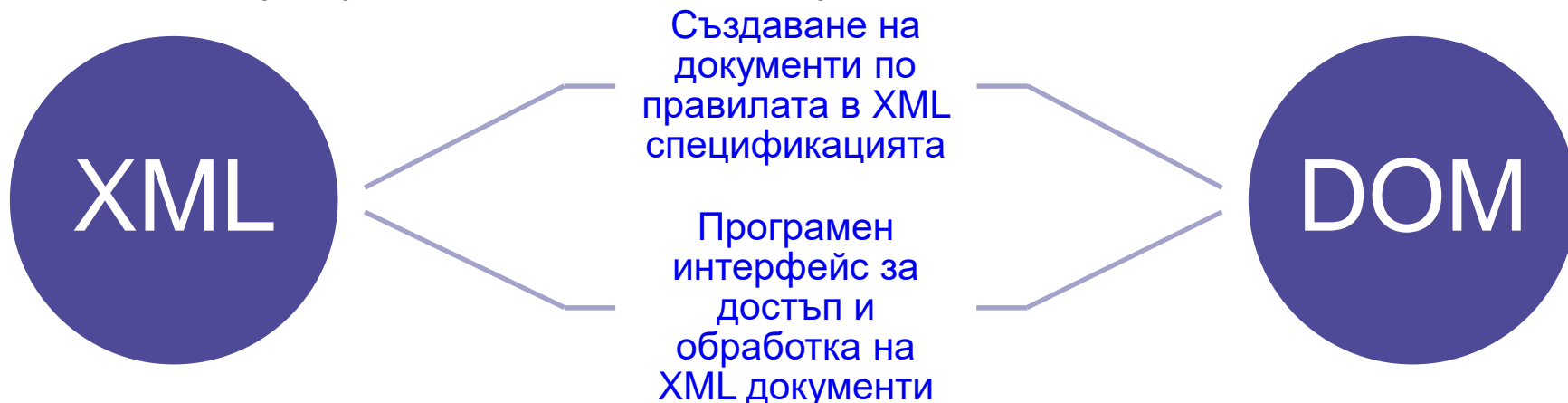
Предимства на DOM 1/2

- Когато правим разбор на XML документ с DOM парсер, получаваме дървовидна структура (наречена **DOM дърво**), която съдържа всички елементи на нашия документ
- DOM предлага разнообразие от функции, които можем да използваме, за да се прегледаме и променяме съдържанието и структурата на документа => удобен за приложни рамки и генератори



Предимства на DOM 2/2

- Стандартизиран начин за достъп до части на XML документ
 - Създаване на документ и части от документи
 - Обхождане на документ
 - Преместване, копиране и премахване на части от документ
 - Добавяне и промяна на атрибути
- Стандартизиран начин за обработка на данни посредством създаване на обектен модел, съответстващ на структурата на XML документа



Как работи DOM?

- DOM обикновено се добавя като междинен слой между XML парсера и приложението
- Изисквания за достъп на приложение до XML документ посредством DOM
 - XML парсер
 - DOM имплементация
 - Съществуват DOM имплементации с вградени XML парсери (например MS XML)

XML
документ

XML парсер

DOM

Приложение

Нива на DOM спецификацията



Level 4

- 2015-11-19



Level 3

- 2004-04-07 (Core)



Level 2

- 2000-11-13 (Core)



Level 1

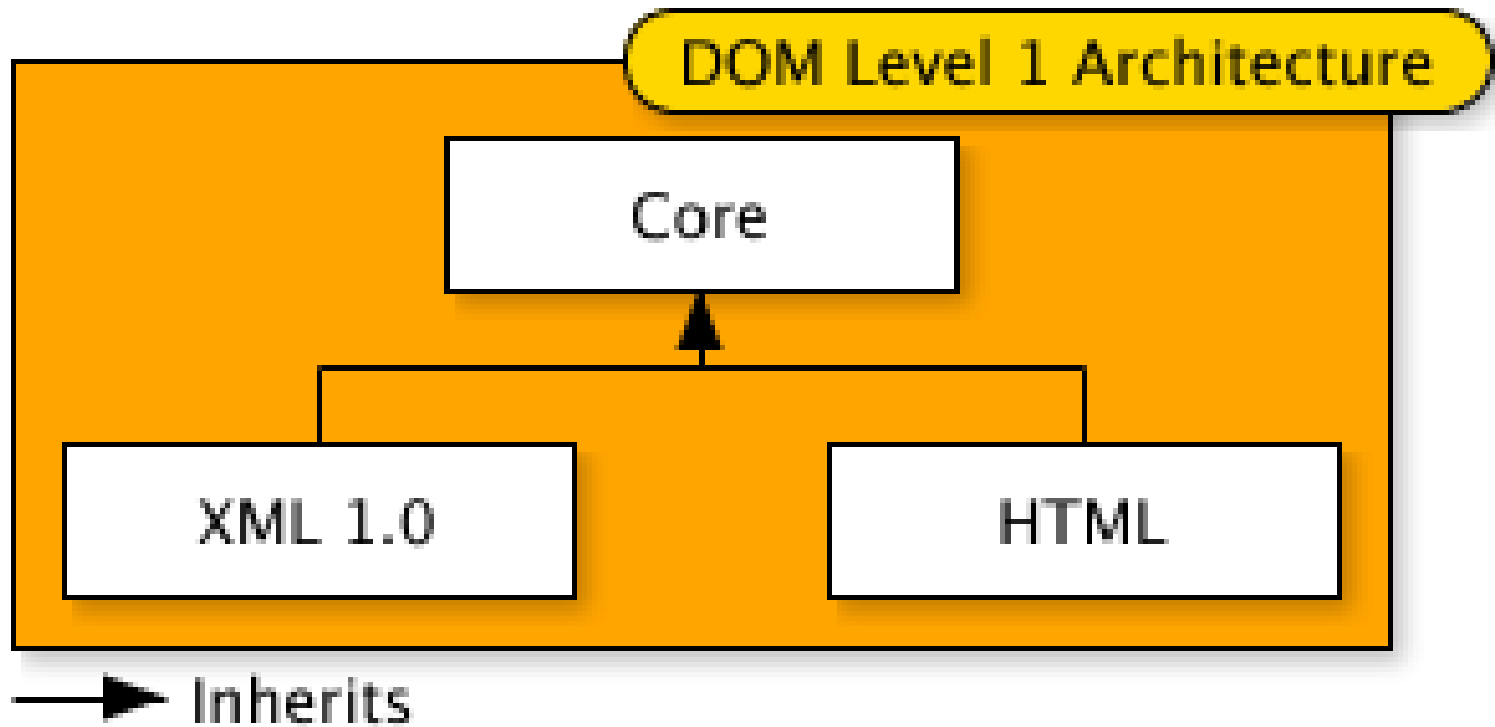
- 1998-10-01

DOM нива



- **DOM Level 1, 2 и 3** са най-използваните спецификации на W3C.
- Спецификацията **DOM Level 2/3/4** е обширен набор от интерфейси, стандартизиращи начина на третиране на парсваните XML данни.
- Моделът на DOM е йерархичен, или дървовиден. Документът може да се обработва отгоре-надолу или обратно, с достъп до всеки междинен възел или листо.
- Спецификацията **DOM Level 2/3/4** е разделена на отделни (под)спецификации

DOM Level 1



DOM Level 2

XML DOM Level 2 под-спецификации са:

DOM Level 2 Core (<http://www.w3.org/TR/DOM-Level-2-Core/>)

DOM Level 2 Views

DOM Level 2 Style

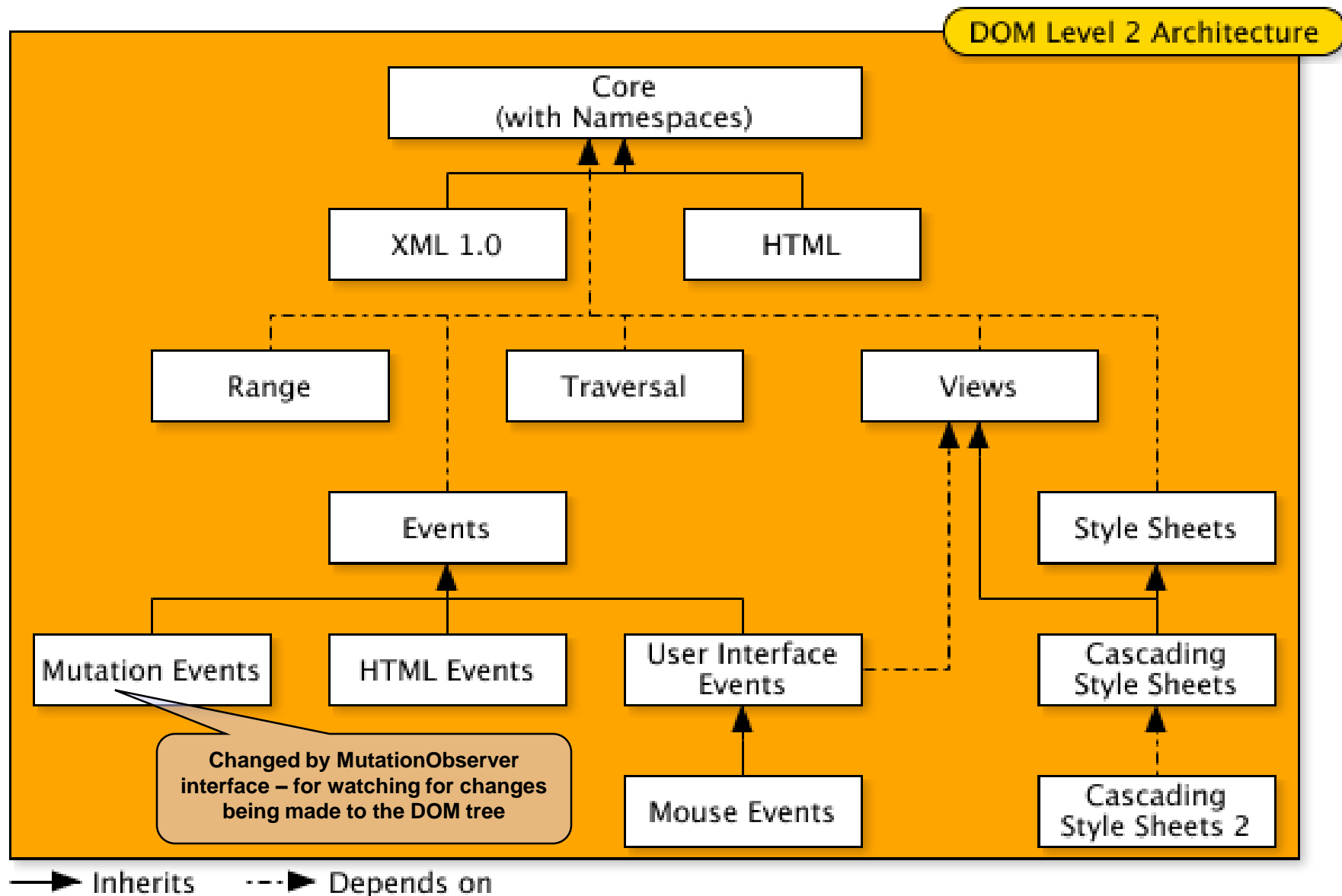
DOM Level 2 Events

DOM Level 2 Traversal-Range

DOM Level 2 HTML (<http://www.w3.org/TR/DOM-Level-2-HTML/>)

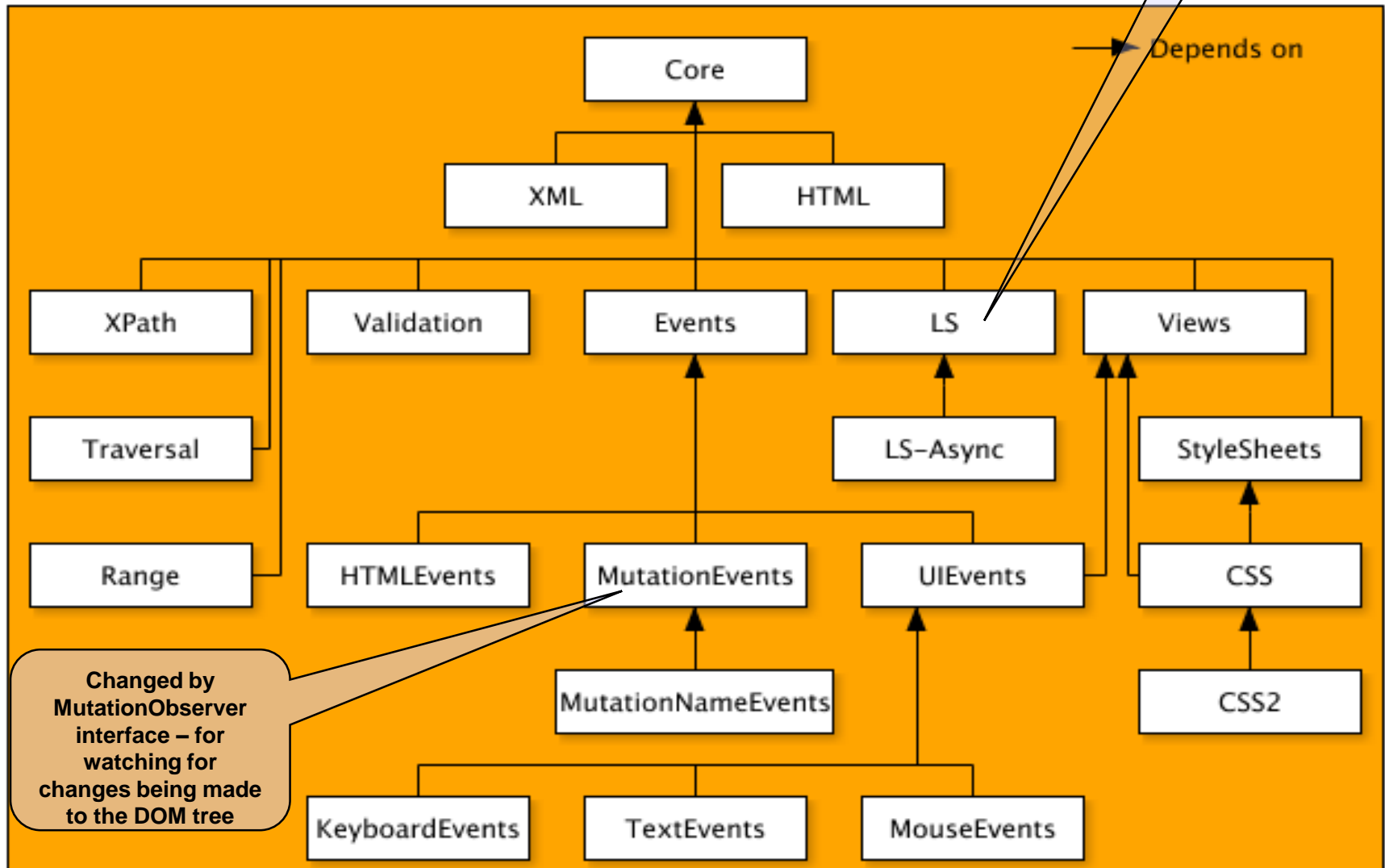
- Повечето приложения, които поддържат DOM Level 2, всъщност поддържат само DOM Level 2 Core спецификацията, т.е. достъп и манипулация на HTML или XML (т.нар. basic parsing)
- DOM Level 2 Core представя документа като йерархия от Node обекти и е платформено и езиково независим интерфейс за динамичен достъп и промяна на съдържанието на документа
- Официален W3C DOM Уеб сайт: <http://www.w3.org/DOM/>

DOM Level 2



DOM Level 3

Load
and
Save



DOM имплементации

- **Xerces**

- Част от Apache проекта, осигурява парсери за Java и C++, имплементира W3C XML и DOM Level 1, 2 (и 3 от Xerces-J 2.7.0 насам) стандартите (<http://xml.apache.org>)

- **4DOM**

- Осигурява на Python разработчиците инструмент за манипулиране на HTML и XML документи

- **ActiveDOM**

- Active-X контрол, който осигурява зареждане и създаване на XML файлове въз основа на W3C DOM 1.0 спецификацията

- **Docuverse DOM**

- Цялостна имплементация на W3C DOM (Document Object Model) API в Java

- **PullDOM и MiniDOM**

- Приложен програмен интерфейс за работа с DOM обекти в Python

- **TcIDOM**

- Език, който свързва DOM със скриптовия език TCL (Tool Command Language)

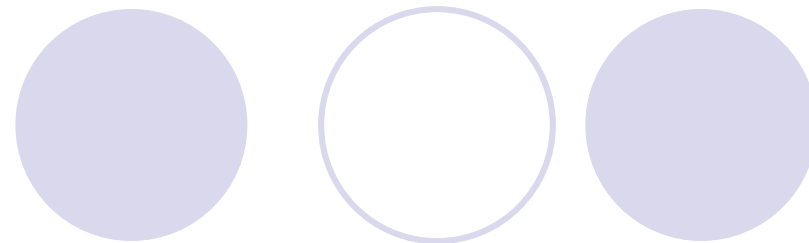
- **XDBM**

- XML Database Manager, осигурен като вградена база от данни за използване в рамките на други приложения посредством DOM базиран приложен програмен интерфейс

DOM стандарт 1/2

- DOM е официален стандарт на www.w3.org
- Използва ОО подход
- Композиран е от различни интерфейси
 - `org.w3c.dom.*`
- Централен клас е '**Document**' (DOM дърво)
- DOMString
 - Спецификация на тип данни, осигуряваща еднакво функциониране на всички DOM имплементации
 - sequence of 16-bit unsigned integers
 - typically interpreted as UTF-16 code units

DOM стандарт 2/2



- DOM ядро
 - Множество от интерфейси за работа с базови документи
- DOM опционални модули
 - Допълнителни интерфейси за работа с други документи като HTML и CSS, които могат да се имплементират при необходимост
- DOM стандартът:
 - включва обхождане на дървото (от DOM Level 2 насам)
 - включва генериране на XML формат (от DOM Level 3 насам) - запазване на DOM дървото като XML документ

DOM опционални модули

- DOM Views

- Осигурява на програмите и скриптовете динамичен достъп и обновяване на документния изглед

- DOM Events

- Осигурява система за управление на събития

- DOM HTML

- Осигурява на програмите и скриптовете динамичен достъп и обновяване на съдържанието и структурата на HTML документи

- DOM CSS

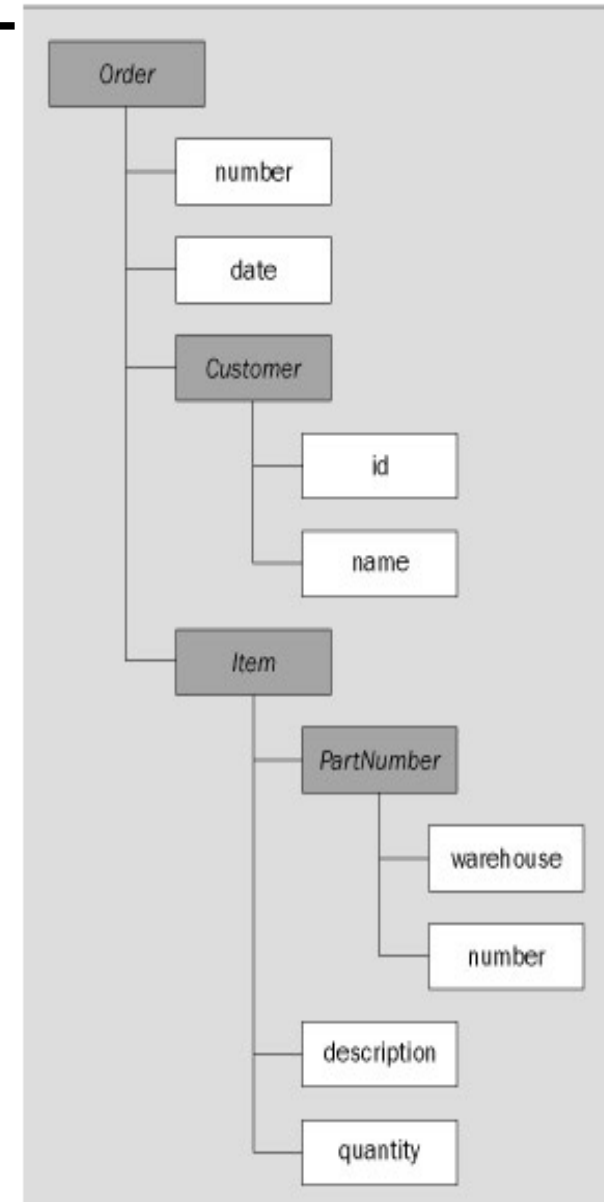
- Осигурява на програмите и скриптовете динамичен достъп и обновяване на съдържанието и структурата на CSS документи

- DOM обхождане и обхват

- Осигурява на програмите и скриптовете динамично обхождане и идентифициране на области в документ

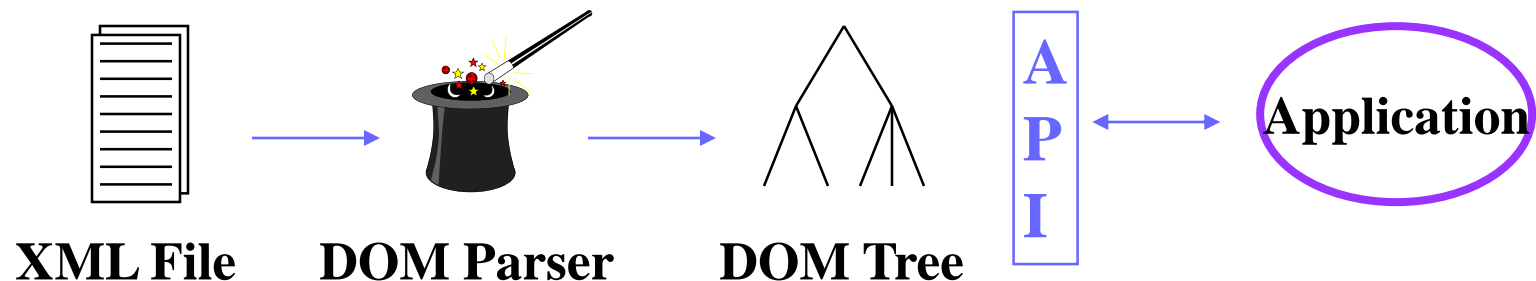
XML документ като обект

```
<?xml version="1.0"?>
<order number="312597">
  <date>2018/11/2</date>
  <customer id="216A">
    Company A
  </customer>
  <item>
    <part-number warehouse="Warehouse 11">
      E16-25A
    </part-number>
    <description>
      Production-Class Widget
    </description>
    <quantity>
      16
    </quantity>
  </item>
</order>
```



Създаване на DOM дърво

- Всяка DOM имплементация има метод за предаване на XML файл към метод-фабрика, който връща обект-екземпляр на Document – представя елемента корен на целия документ.
- След това използваме стандартния DOM интерфейс за интеракции с XML структурата




Създаване на DOM дърво - пример

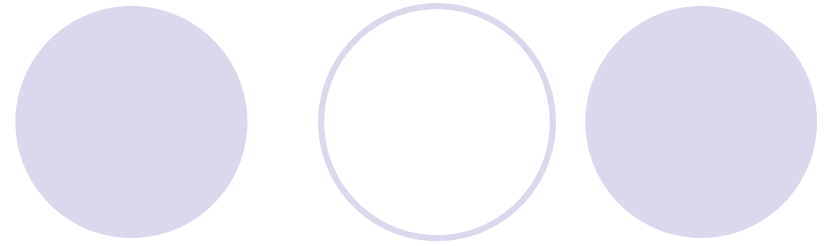
```
import org.w3c.dom.*;           //DOM interfaces
import com.sun.xml.tree.*;      //Using Sun classes
import org.xml.sax.*;           //Need SAX classes

public class myClass {
...
Document myDoc; //Document object
try {
    //if 'true' -> validate the XML!
    myDoc =
        XmlDocument.createXmlDocument("file:/doc.xml", true);
} catch (IOException err) {...}
   catch (SAXException err) {...}
   catch (DOMException err) {...}

//If no exceptions, should have a 'Document' object
XML DOM
```



DOM интерфейси



- Възли
 - Вътрешно представяне на обектите в дървовидна структура, съхранена в паметта
- Интерфейси
 - Осигуряват механизъм за управление на обектите

```
<parent>  
  <child id="123">  
    text goes here  
  </child>  
</parent>
```

Document Node (Document Root)

NodeList

Element Node (<parent>)

NodeList

Element Node (<child>)

NamedNodeMap

NodeList

Attr Node (id="123")

Text CharacterData
Node (text goes here)

DOM дърво на XML документ

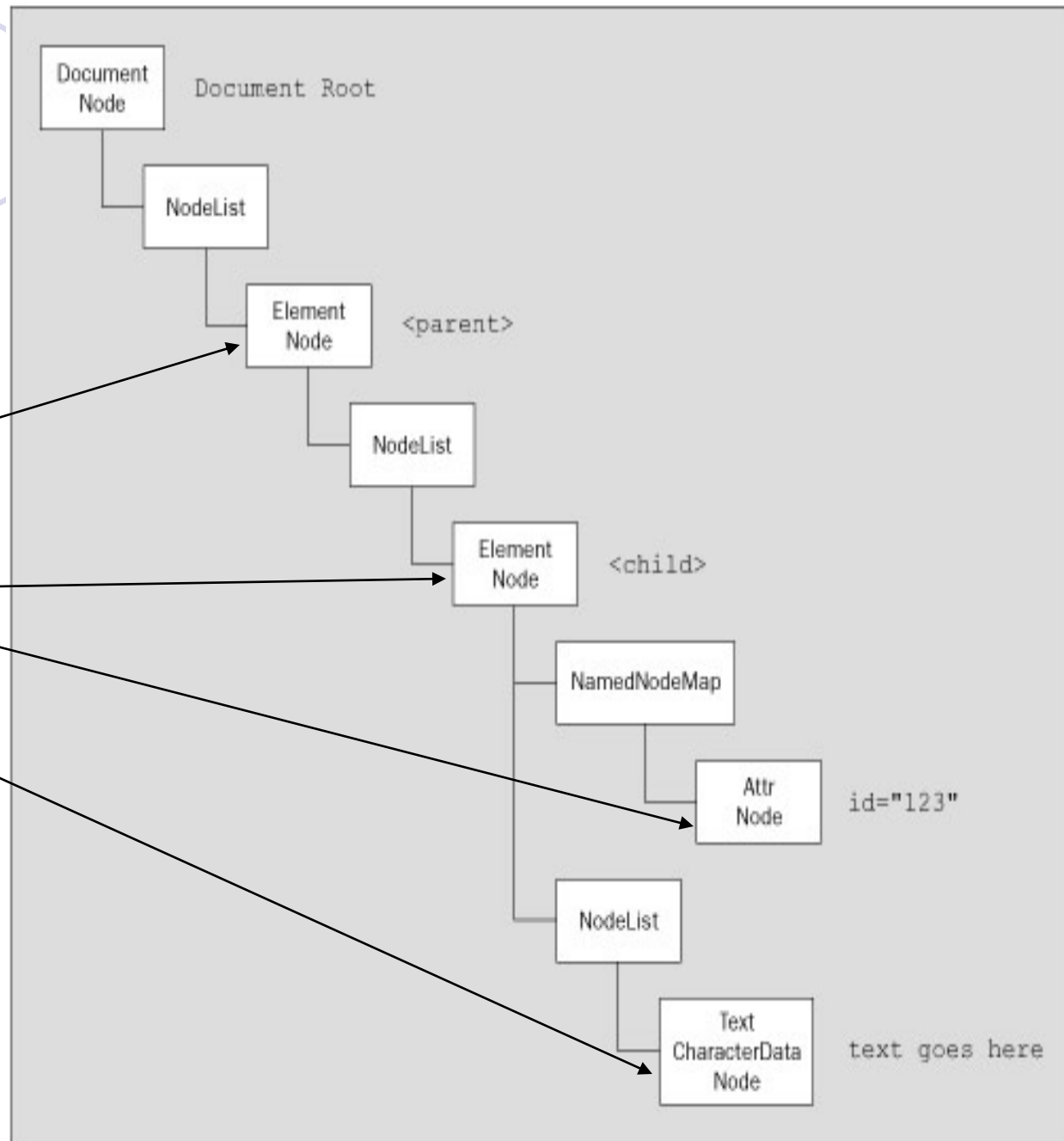
<parent>

<child id="123">

text goes here

</child>

</parent>



Структурен модел 1/2

- Елементи в XML документа
 - Елемент <parent>
 - Елемент <child>
 - Атрибут id на елемент <child>
 - Съдържание на елемент <child>
- Възел Document в DOM
 - Дефинира контекст на документа
 - Имплементира методи за създаване на обекти в документа
 - Поддържа интерфейс Document
 - Притежава само един дъщерен елемент
 - Може да притежава други XML елементи като коментари, инструкции за обработка, декларации на типове

Структурен модел 2/2

- Възел NodeList
 - Дефинира списък от възли Nodes на едно ниво в дървото
 - Поддържа интерфейс NodeList
 - Добавя се автоматично преди елемент
- Възел NamedNodeMap
 - Дефинира неопределено множество от възли, реферирани по име на атрибут
 - Добавя се автоматично преди атрибути

DOM имплементации

- Съществуват различни имплементации на DOM
- Поради това DOM предоставя **DOM Core** – множество от основни интерфейси
- Допълнително, DOM предоставя множества от интерфейси за други формати – **DOM HTML**, **DOM CSS**, и др. Напр. **DHTML** използва обекти от *DOM HTML* множеството.
- *Всяка имплементация предоставя класове, имплементиращи интерфейсите*

DOM + HTML = DHTML

- Комбинацията от DOM (с HTML API), скриптови езици и HTML изгражда т.нар. **Dynamic HTML** (или **DHTML**), където съдържанието на HTML документа се представя чрез обектен модел. *Когато Уеб страница се зареди в DHTML браузър, обекти се създават за всеки елемент от страницата. Това прави възможно изпълнението на скриптове в страницата, обръщащи се към методи и свойства (properties) на тези обекти.*
- Например, за HTML формата:
 - `<form id="frmScratchForm" name="frmScratchForm">`
 - `<input name="rdoRadio" type="radio" checked>First
`
 - `<input name="rdoRadio" type="radio">Second`
 - `</form>`
- JavaScript код може да избере втория радио бутон (индексът започва от 0) така:
`document.frmScratchForm.rdoRadio[1].checked = true;`
- Така след зареждане на страницата стойностите на радио-бутоните могат да се променят програмно и *динамично*.

Примерен тест с ползване на MSXML

```
<html>
  <head><title>DOM Demo</title>
  <script language="JavaScript">
    var oDOM;
    oDOM = new
      ActiveXObject("MSXML.DOMDocument");
    oDOM.async = true;
    oDOM.load("domnode.xml");
    //our code will go here...
  </script>
</head>
<body>
  <p>This page demos some of the DOM capabilities.</p>
</body>
</html>
```

Документът ще се зареди асинхронно, т.е. методът load() ще върне управлението веднага, а документът ще продължи да се зарежда във фонов режим.

Основни типове интерфейси

DOM ядро

Базови интерфейси

Задължителни за всички
имплементации на DOM

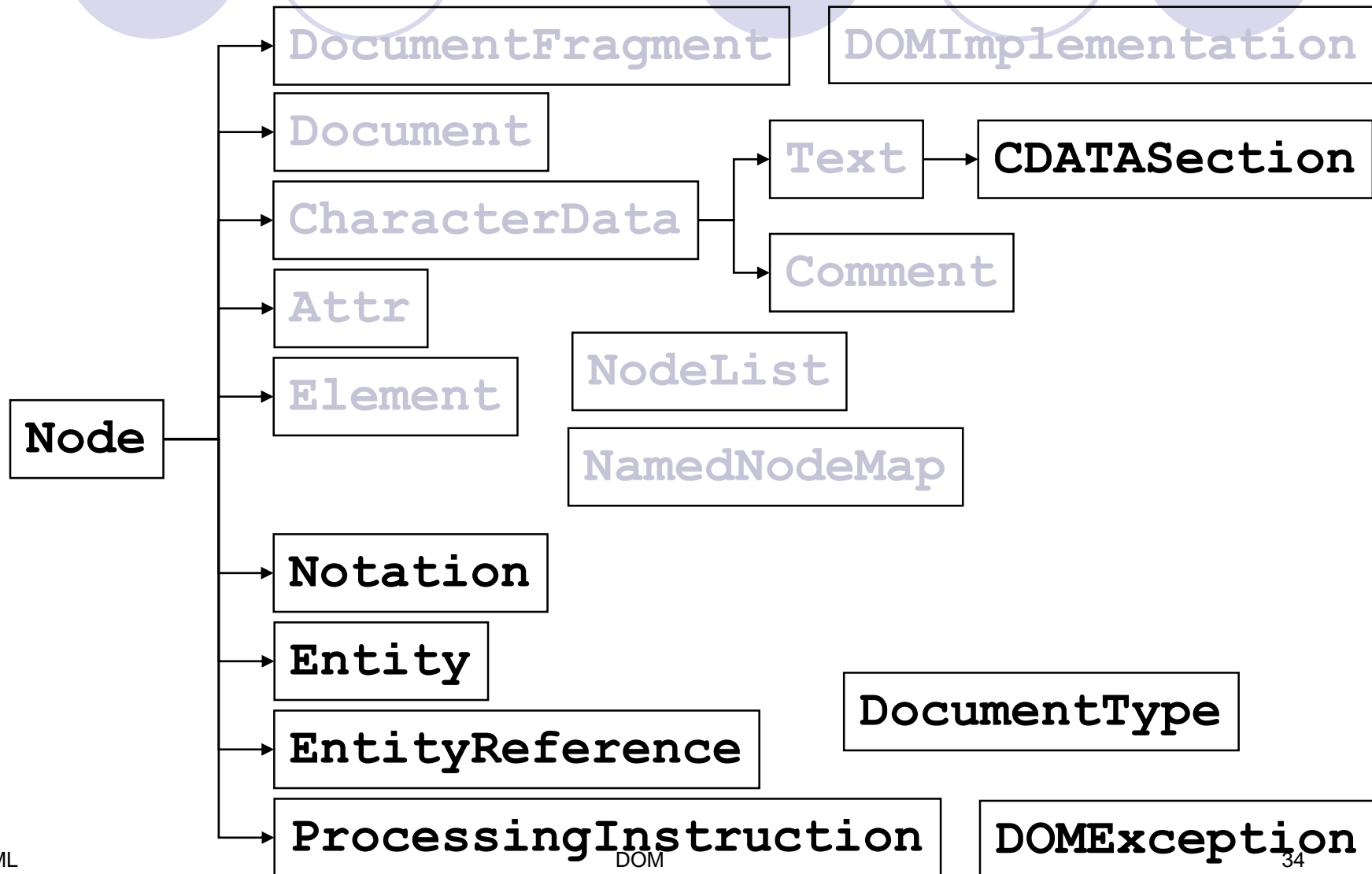
Разширени интерфейси

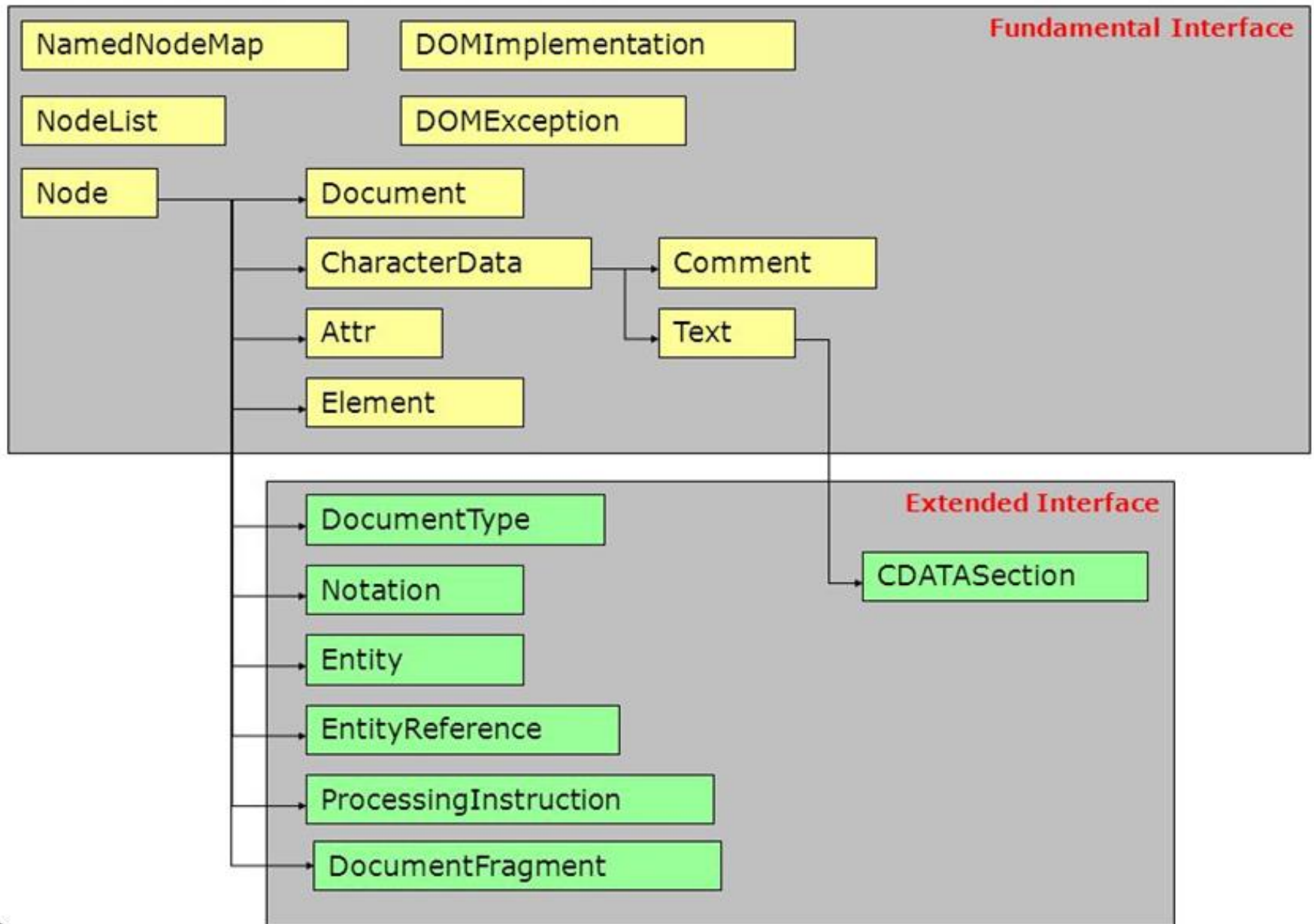
Необходими са само за DOM
имплементации, работещи с XML

Базови (фундаментални) DOM интерфейси

Node	Document	DOMImplementation
DocumentFragment	NodeList	Element
NamedNodeMap	Attr	CharacterData
Text	Comments	DOMException

Разширени DOM интерфейси



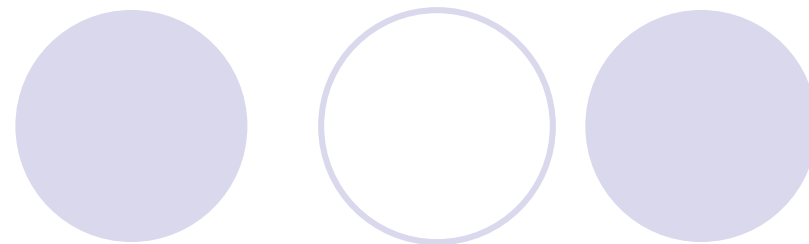


DOM интерфейси

- DOM дефинира няколко главни интерфейса:

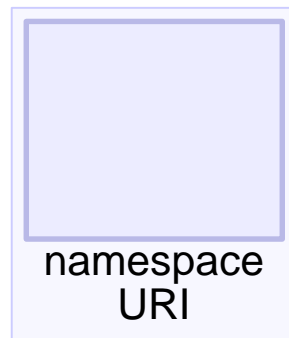
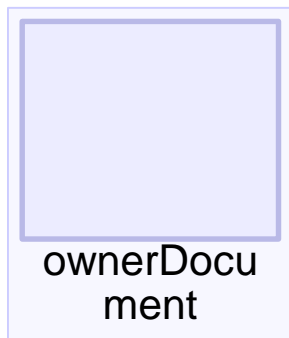
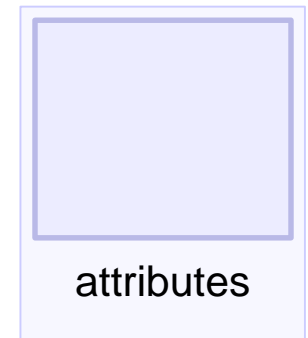
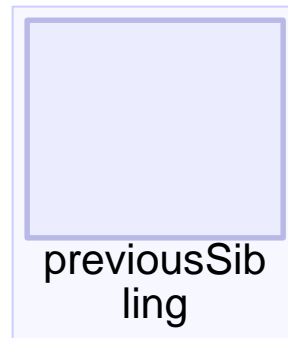
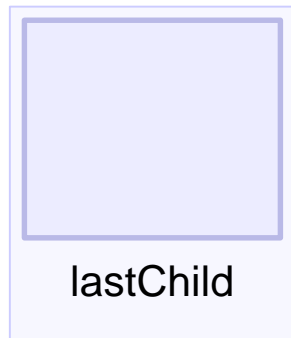
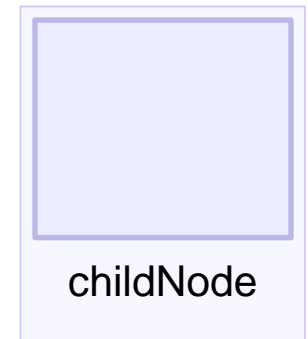
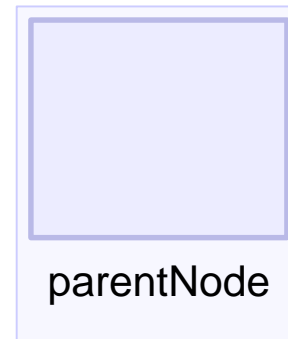
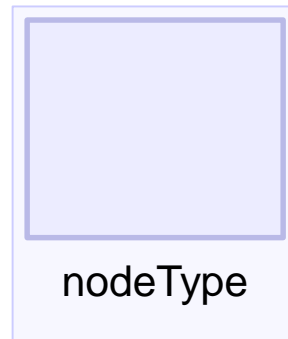
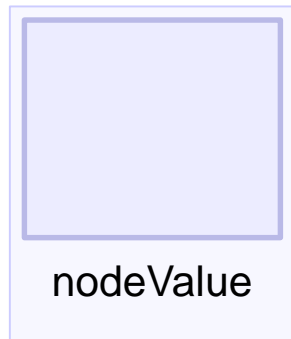
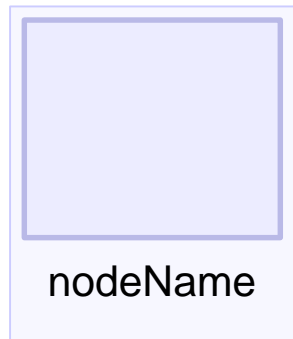
Node	Базов тип данни на DOM
Element	Представя елемент
Attr	Представя атрибут на елемент
Text	Представя съдържанието на атрибут или елемент
Document <small>XML</small>	Представя целия XML документ; Document обектът често се означава като DOM дърво. <small>DOM</small>

Интерфейс NODE



- Всяка част от XML документа се представя с интерфейс NODE
- Функционалност
 - Обхождане на дървото: всяка обработка изисква позициониране на определено място в дървото
 - Получаване на информация за възел: тип, атрибути, име и стойност
 - Добавяне, премахване и обновяване на възли
 - Промяна на структурата на дървото

Свойства на интерфейс Node



Методи на интерфейс Node

Методи

- insertBefore(newChild, refChild)
- replaceChild(newChild, oldChild)
- removeChild(oldChild)
- appendChild(newChild)
- hasChildNodes()
- cloneNode(deep)
- normalize()
- supports(feature, version)

Методи на Node 1/2

- Три категории от методи:
 - Характеристики на Node
 - name, type, value
 - Контекстна локация и достъп до съседни възли:
 - parents, siblings, children, ancestors, descendants
 - Модификации на Node:
 - Edit, delete, re-arrange child nodes

Методи на Node 2/2

```
short      getNodeType() ;

String     getNodeName() ;

String     getNodeValue()
throws DOMException;

void       setNodeValue(String value)
throws DOMException;

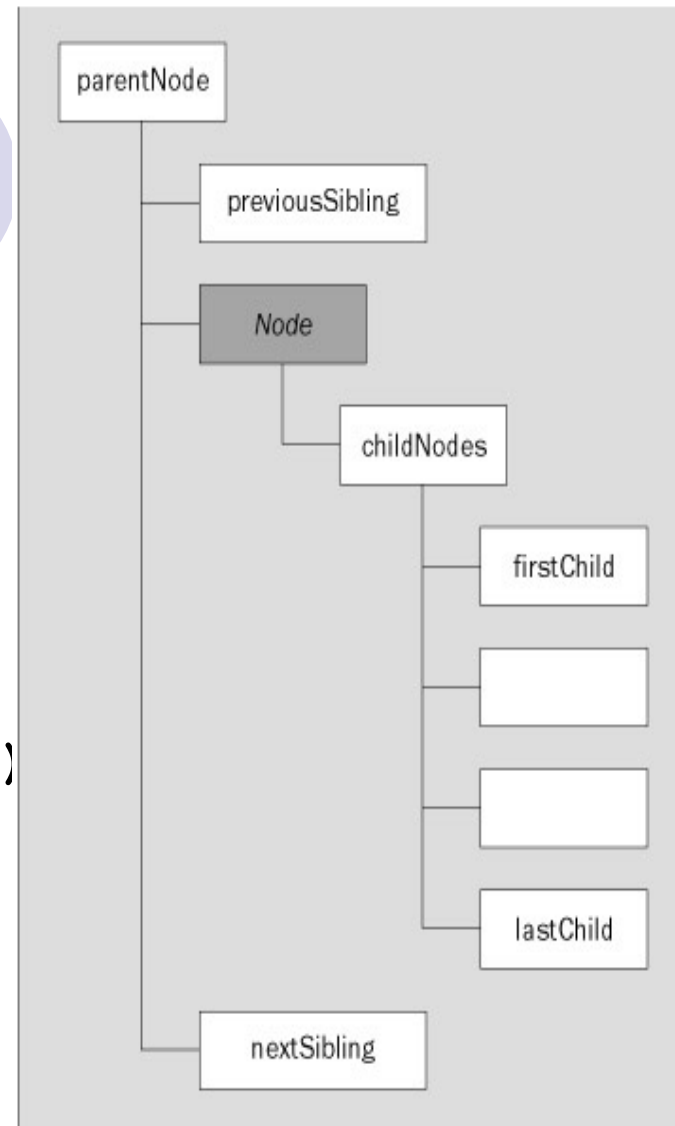
boolean    hasChildNodes() ;

NamedNodeMap getAttributes() ;
//returns unordered collection of nodes

Document   getOwnerDocument() ;
```

XML

DOM



Прилага се само, ако възелът е елемент с атрибути
Връща като резултат NamedNodeMap колекция с атрибутите на възела или NULL

Типове възли - `getNodeTypes()`

<code>ELEMENT_NODE</code>	<code>= 1</code>	<code>PROCESSING_INSTRUCTION_NODE</code>	<code>= 7</code>
<code>ATTRIBUTE_NODE</code>	<code>= 2</code>	<code>COMMENT_NODE</code>	<code>= 8</code>
<code>TEXT_NODE</code>	<code>= 3</code>	<code>DOCUMENT_NODE</code>	<code>= 9</code>
<code>CDATA_SECTION_NODE</code>	<code>= 4</code>	<code>DOCUMENT_TYPE_NODE</code>	<code>= 10</code>
<code>ENTITY_REFERENCE_NODE</code>	<code>= 5</code>	<code>DOCUMENT_FRAGMENT_NODE</code>	<code>= 11</code>
<code>ENTITY_NODE</code>	<code>= 6</code>	<code>NOTATION_NODE</code>	<code>= 12</code>

Пример:

```
if (myNode.getNodeTypes() == Node.ELEMENT_NODE) {  
    //myNode.getNodeTypes() returns short!!  
    //process node  
    ...  
}
```

Node имена и стойности

- Всеки възел има име ... и евентуално - стойност
- Името не е уникален идентификатор (а само локация)

Тип	Име на интерфейс	Име	Стойност
ATTRIBUTE_NODE	Attr	<i>Attribute name</i>	<i>Attribute value</i>
DOCUMENT_NODE	Document	#document	<i>NULL</i>
DOCUMENT_FRAGMENT_NODE	DocumentFragment	#document-fragment	<i>NULL</i>
DOCUMENT_TYPE_NODE	DocumentType	<i>DOCTYPE name</i>	<i>NULL</i>
CDATA_SECTION_NODE	CDATASection	#cdata-section	<i>CDATA content</i>
COMMENT_NODE	Comment	<i>Entity name</i>	<i>Content string</i>
ELEMENT_NODE	Element	<i>Tag name</i>	<i>NULL</i>
ENTITY_NODE	Entity	<i>Entity name</i>	<i>NULL</i>
ENTITY_REFERENCE_NODE	EntityReference	<i>Entity name</i>	<i>NULL</i>
NOTATION_NODE	Notation	<i>Notation name</i>	<i>NULL</i>
PROCESSING_INSTRUCTION_NODE	ProcessingInstruction	<i>Target string</i>	<i>Content string</i>
TEXT_NODE	Text	#text	<i>Text string</i>

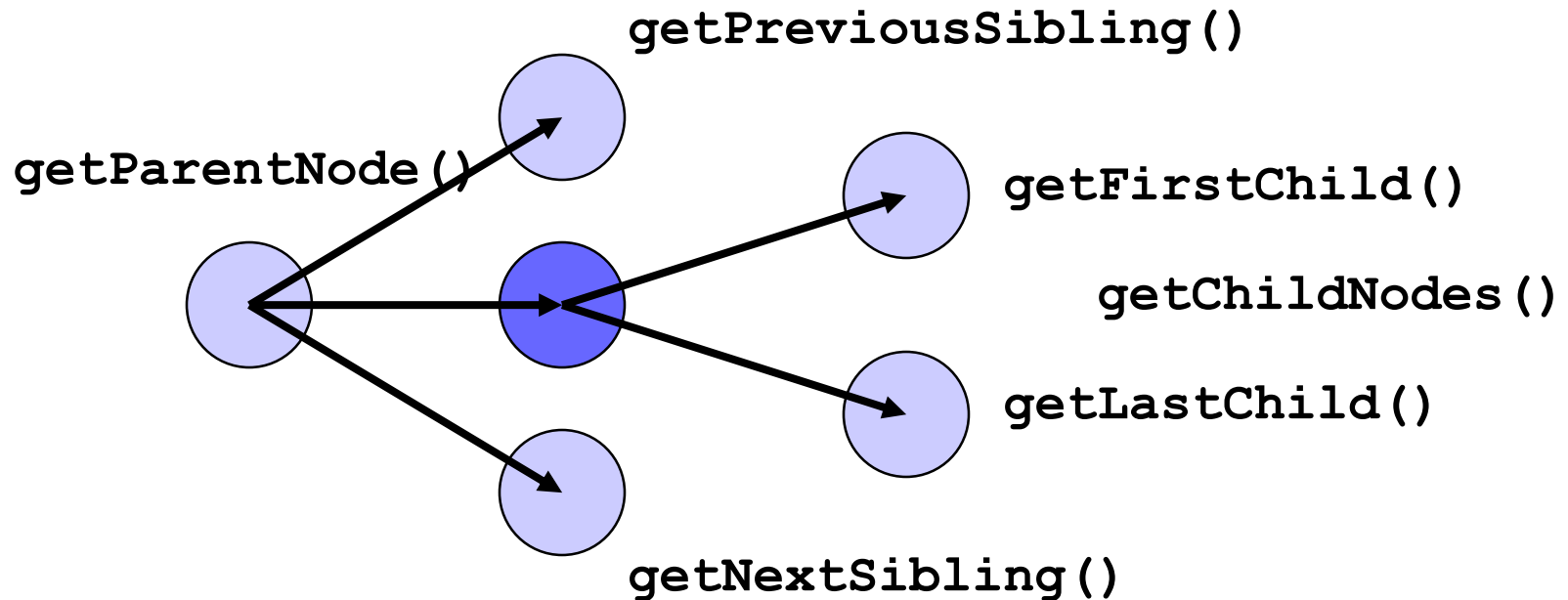
Възли-деца (Child Nodes)

- Повечето **Nodes** не могат да имат деца, с изключение на:
 - **Document, DocumentFragment, Element**
- Проверка за съществуване на деца:
 - ```
if (myNode.hasChildNodes()) {
 //process children of myNode
 ...
}
```

# Node навигация 1/2

- Всеки възел има специфично местоположение в дървото
- Node интерфейсът има методи за намиране на възлите от обкръжението:
  - Node    get**First**Child() ;
  - Node    get**Last**Child() ;
  - Node    get**Next**Sibling() ;
  - Node    get**Previous**Sibling() ;
  - Node    get**Parent**Node() ;
  - NodeList    get**Child**Nodes() ;

# Node навигация 2/2



```
Node parent = myNode.getParentNode();
if (myNode.hasChildren()) {
 NodeList children = myNode.getChildNodes();
```

# Пример

- For domnode.xml:

```
<root>
 <DemoElement
 DemoAttribute="stuff">
This is the PCDATA
 </DemoElement>
</root>
```

```
<script language="JavaScript">
```

```
var oDOM;
oDOM = new
```

```
 ActiveXObject("MSXML.DOMDocument");
```

```
oDOM.async = false;
```

```
oDOM.load("domnode.xml");
```

```
//our code will go here...
```

```
alert(oDOM.firstChild.firstChild.firstChild.nodeValue);
```

```
</script>
```

XML

DOM



# DOM дърво

`alert(oDOM.firstChild.firstChild.firstChild.nodeValue)`

- За domnode.xml:

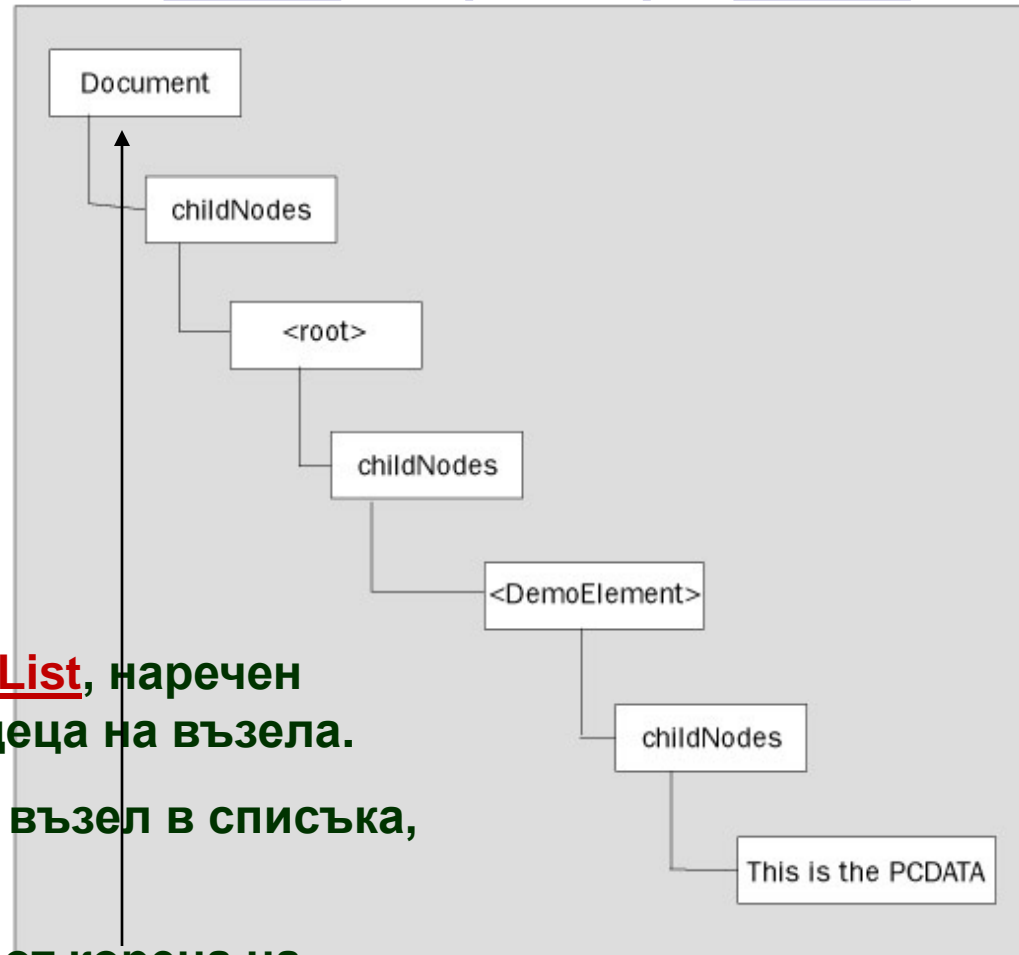
`<root>`

`<DemoElement  
DemoAttribute="stuff">`

`This is the PCDATA`

`</DemoElement>`

`</root>`



- Всеки Node обект предоставя NodeList, наречен childNodes, който съдържа всички деца на възела.
- Имаме директен достъп до първия възел в списъка, посредством firstChild property.
- В предходния код се придвижваме от корена на документа към елемента `<DemoElement>` посредством firstChild property.

DOM

`oDOM.firstChild.firstChild.firstChild.nodeValue`



# Node манипулации 1/2

- Децата на възел в DOM дърво могат да бъдат манипулирани – добавяни, редактирани, изтривани, копирани, премествани и т.н.

`Node removeChild(Node old)` throws `DOMException`;

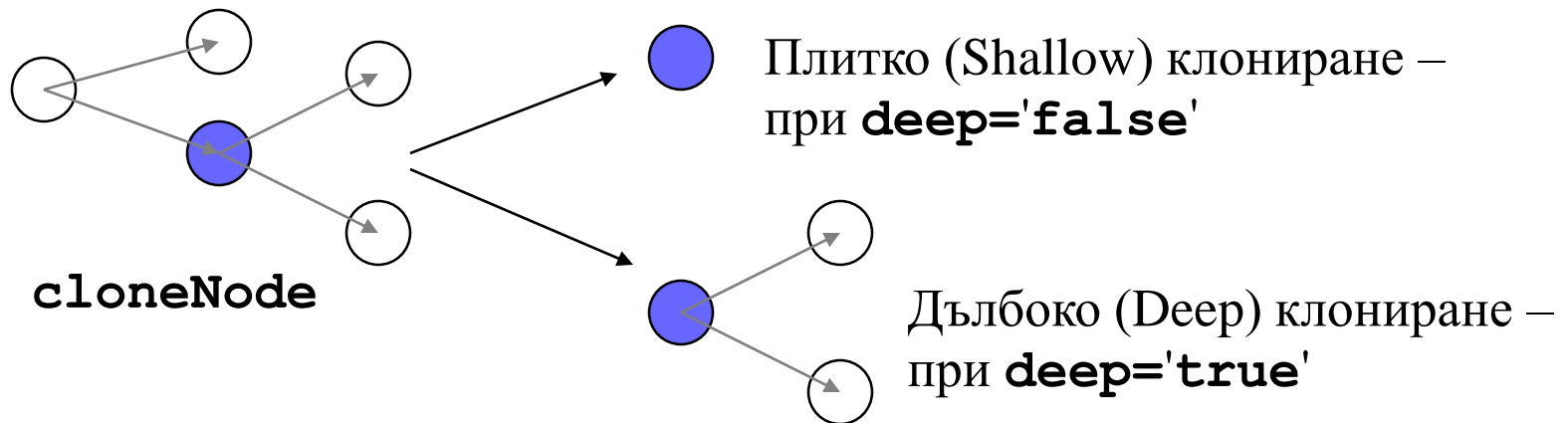
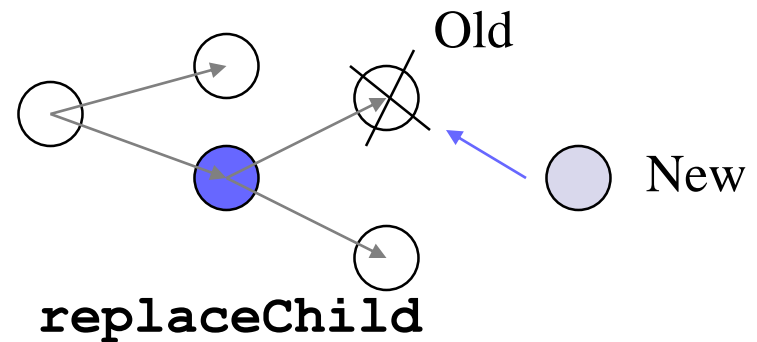
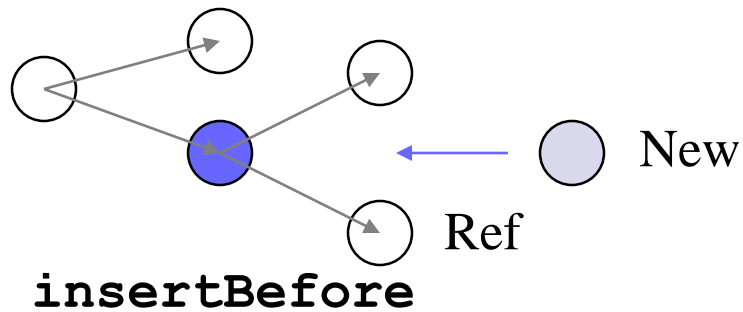
`Node insertBefore(Node new, Node ref)` throws `DOMException`;

`Node appendChild(Node new)` throws `DOMException`;

`Node replaceChild(Node new, Node old)` throws `DOMException`;

`Node cloneNode(boolean deep) ;`

# Node манипулации 2/2



# Добавяне и премахване на възли

- Методът `appendChild`

```
objParentNode.appendChild(objChildNode) ;
```

- Методът `insertBefore`

- Ако не се укаже възел, обозначаващ място на вмъкване, то новият наследник се добавя в края на списъка
- Ако възелът, който се добавя, вече съществува, то се извършва подмяна

```
objParentNode.insertBefore(objChildNode,
 objParentNode.lastChild) ;
```

- Методът `removeChild`

- Изтритият възел се връща като резултат в случай, че е необходима последваща обработка

```
objOldChild =
 objParent.removeChild(objParent.lastChild) ;
```

# Замяна и клониране на възли

## ● Методът `replaceChild`

```
objOldChild = objParent.replaceChild(objNewChild,
 objParent.firstChild);
```

## ● Методът `cloneNode`

- Клонираните възли могат да се използват само в документа, в който са създадени

```
<name id="1">John</name>
```

```
objNewNode = objNode.cloneNode(false);
```

```
//objNewNode is now <name id="1"/>
```

false - Default. Clone only the node and its attributes

```
objNewNode = objNode.cloneNode(true);
```

```
//objNewNode is now <name id="1">John</name>
```

true - Clone the node, its attributes, *and* its descendants

# Document::Node интерфейс 1/2

- Представя целия XML документ (корен на дървото)
- Наследява интерфейса Node, като добавя функционалност за обхождане на DOM дървото
- Възелът Document обхваща всички възли в DOM дървото, включително и кореновия елемент
- Методи:

*//Information from DOCTYPE - See 'DocumentType'*

DocumentType           getDocumentType();

*//Information about capabilities of DOM implementation*

DOMImplementation   getImplementation();

*//Returns reference to root node element*

Element                getDocumentElement();

*//Searches for all occurrences of 'tagName' in nodes*

NodeList               getElementsByTagName(String tagName);

*//Търси елемент по атрибут ID и връща NodeList*

NodeList               getElementsByTagName(String ID);

# Document::Node интерфейс 2/2

- Метод-фабрики за създаване на възли:

Element **createElement**(String tagName) throws DOMException;

DocumentFragment **createDocumentFragment**();

Text **createTextNode**(String data);

Comment **createComment**(String data);

CDATASection **createCDATASection**(String data) throws  
DOMException;

ProcessingInstruction **createProcessingInstruction**(  
String target, String data) throws DOMException;

Attr **createAttribute**(String name) throws DOMException;

EntityReference **createEntityReference**(String name)  
XML DOM throws DOMException;<sup>54</sup>

# Създаване на XML документи

- Създаване на възел

```
var objNode, objText;
objNode = objDOM.createElement("root");
objText = objDOM.createTextNode("root PCDATA");
objDOM.appendChild(objNode);
objNode.appendChild(objText);
```

- Създаване на атрибут

```
var objAttr;
objAttr = objDOM.createAttribute("id");
//set the attribute's value
objAttr.nodeValue = "123";
//append the attribute to the element
objNode.attributes.setNamedItem(objAttr);
alert(objDOM.xml);
```

```
<root id="123">root PCDATA<root>
```

# Element::Node интерфейс 1/2

- Две категории методы:

- Общи методы за елемента:

```
String getTagName () ;
NodeList getElementsByTagName () ;
Void normalize () ;
// removes empty Text nodes, and joins adjacent Text nodes.
```

- Методи за управление на атрибути:

```
String getAttribute (String name) ;
void setAttribute (String name, String value)
 throws DOMException;
void removeAttribute (String name)
 throws DOMException;
Attr getAttributeNode (String name) ;
void setAttributeNode (Attr new)
 throws DOMException;
void removeAttributeNode (Attr old)
 throws DOMException;
```

разлика



# Element::Node интерфейс 2/2

- Само **Element** обектите могат да имат атрибути, като атрибутните методи на **Element** са прости:
  - Изискват име на атрибута
  - Не различават стойност по подразбиране (определена в DTD) от зададена в XML файла
  - Не могат да определят типа на атрибута [String]
- Вместо тях използвайте метода **getAttributes()** на **Node**
  - Връща **Attr** обекти като **NamedNodeMap**

# Интерфейс element: Пример 1/2

```
<?xml version="1.0"?>
```

```
<root first='John' last='Doe' />
```

```
var objDOM;
```

```
objDOM = new ActiveXObject("MSXML2.DOMDocument");
```

```
objDOM.async = false;
```

```
objDOM.load("ch06_ex8.xml");
```

```
var objElement;
```

```
objElement = objDOM.documentElement;
```

```
...
```

```
alert(objElement.getAttribute("first"))
```

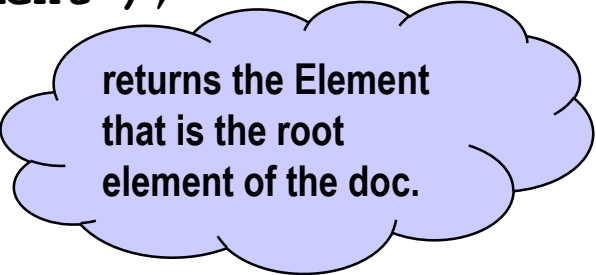
```
objElement.setAttribute("first", "Bill");
```

```
var objAttr;
```

```
objAttr = objElement.getAttributeNode("first");
```

```
objAttr.nodeValue = "Bill";
```

```
objElement.getAttributeNode("first").nodeValue = "Bill";
```



returns the Element  
that is the root  
element of the doc.

# Интерфейс element: Пример 2/2

```
var objElement;
objElement = objDOM.documentElement;
...
var objAttr;
objAttr = objDOM.createAttribute("middle");
objAttr.nodeValue = "Fitzgerald Johansen";
objElement.setAttributeNode(objAttr);
alert(objDOM.xml);
```



```
objElement.setAttribute("middle", "Fitzgerald Johansen");
```

```
<?xml version="1.0"?>
```

```
<root first='John' last='Doe' />
```

```
<?xml version="1.0"?>
```

```
<root first='John'
```

```
 last='Doe'
```

```
 middle='Fitzgerald Johansen' />
```



# Attr::Node интерфейс 1/2

- Интерфейс към обекти, съдържащи атрибутни данни
- Свойство ownerElement //deprecated in DOM 4
  - Връща като стойност елемент, на който атрибутът принадлежи
- Свойства name и value
  - Имат стойности аналогични на тези за nodeName and nodeValue
- Свойство specified
  - Определя дали атрибутът е физически специфициран или е скрит със стойност по подразбиране

//Get name of attribute

```
String getName();
```

//Get value of attribute

```
String getValue();
```

//Change value of attribute

```
void setValue(String value);
```

//if 'true' - attribute defined in element, else in DTD

XML

DOM

```
boolean getSpecified();
```

## Attr::Node интерфейс 2/2

- Атрибутите не са пряка част от дървовидната структура на документа
  - `parentNode`, `previousSibling` and `nextSibling` връщат **null** за **Attr** обект
- Създаваме атрибутни обекти чрез метод-фабрика на **Document**

//Create the empty Attribute node

```
Attr newAttr = myDoc.createAttribute("status");
```

//Set the value of the attribute

```
newAttr.setValue("secret");
```

//Attach the attribute to an element

```
myElement.setAttributeNode(newAttr);
```

# Интерфейсите CharacterData и Text

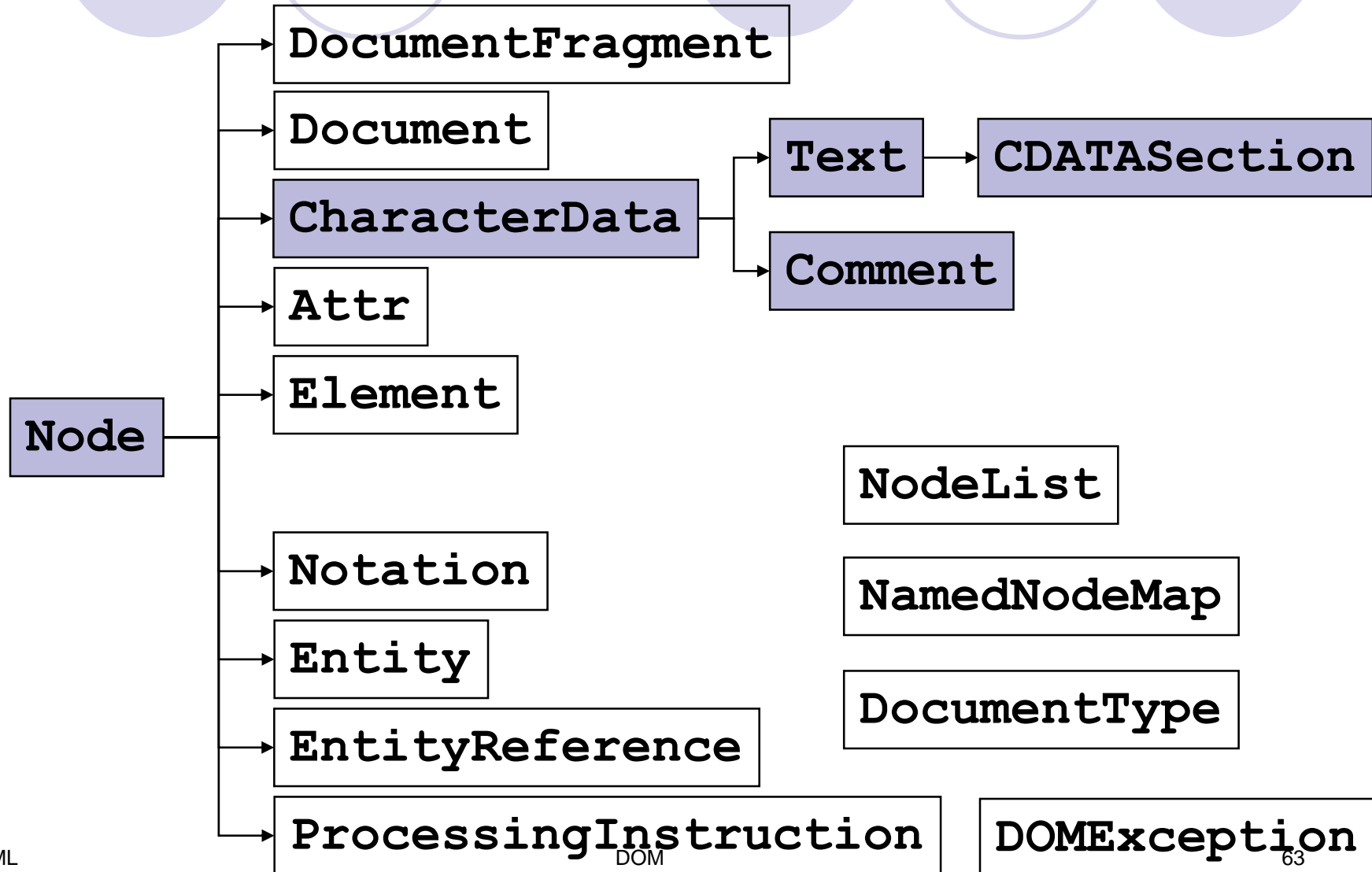
## CharacterData

- Осигурява свойства и методи за работа с текст

## Text

- Разширява CharacterData и се прилага специфично за PCDATA

# Разширени DOM интерфейси



# CharacterData::Node интерфейс

- Полезни общи методи за текстообработка
- Не се използват директно...
  - ... а в наследяващите го типове на възли Text и Comment

```
String getData() throws DOMException;
void setData(String data) throws DOMException;
int getLength();
void appendData(String data) throws DOMException;
String substringData(int offset, int length)
 throws DOMException;
void insertData(int offset, String data)
 throws DOMException;
void deleteData(int offset, int length)
 throws DOMException;
void replaceData(int offset, int length, String data)
 throws DOMException;
```



# Text::Node интерфейс

- Представя текстовото съдържание на **Element** или **Attr**
  - Деца на тези възли
- Винаги са възли-листа (leaf nodes)
- Към **CharacterData** е добавен един метод:
  - **Text splitText(int offset) throws DOMException**
- Метод-фабрика в **Document** за създаване - **createTextNode()**
- Извикването на **normalize()** върху даден **Element** слива **Text** обектите (те може да са повече от един!)

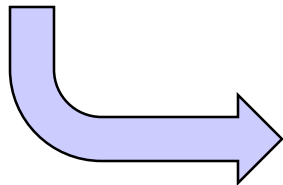
# Comment::Text интерфейс

- Представя коментари
- Автоматично форматиране на коментарите с разделители '<!--' и '-->'
- Няма добавен методи към **CharacterData**
- Метод-фабрика в **Document** за създаване
  - `Comment newComment =  
myDoc.createComment(" my comment ");  
//Note spaces`

[https://www.w3schools.com/xml/met\\_document\\_createcomment.asp](https://www.w3schools.com/xml/met_document_createcomment.asp)

```
function myFunction(xml) {
 var x, i, newComment, xmlDoc, txt;
 xmlDoc = xml.responseXML;
 txt = "";
 x = xmlDoc.getElementsByTagName("book");
 for (i = 0; i < x.length; i++) {
 newComment = xmlDoc.createComment("Revised April 2018");
 x[i].appendChild(newComment);
 }
 for (i = 0; i < x.length; i++) {
 txt += x[i].getElementsByTagName("title")[0].childNodes[0].nodeValue +
 " - " +
 x[i].lastChild.nodeValue + "
";
 }
 document.getElementById("demo").innerHTML = txt;
}
```

XML



Everyday Italian - Revised April 2018  
Harry Potter - Revised April 2018  
XQuery Kick Start - Revised April 2018  
Learning XML - Revised April 2018

# NodeList интерфейс

- Съдържа колекция от ordered Node обекти
- Два метода:

```
//Find number of Nodes in NodeList
```

```
int getLength();
```

```
//Return the i-th Node
```

```
Node item(int index);
```

-----

```
Node child;
```

```
NodeList children = element.getChildNodes();
```

```
for (int i = 0; i < children.getLength(); i++) {
```

```
 child = children.item(i);
```

```
 if (child.getNodeType() == Node.ELEMENT_NODE) {
```

```
 System.out.println(child.getNodeName());
```

```
 }
```

```
}
```

# Интерфейс **NamedNodeMap**

- Дефинира неподредена колекция от възли
- Метод `getNamedItem`
  - Получава като параметър име на възел, например име на атрибут
  - Връща като стойност Node обект
- Метод `removeNamedItem`
  - Получава като параметър име на възел
  - Връща като стойност изтрития Node обект
- Метод `setNamedItem`
  - Получава като параметър възел за добавяне или замяна (ако такъв вече съществува)
- Свойство `length`
  - Връща като стойност броя на елементите в колекцията
- Метод `item`
  - Получава като параметър индекс на възел

# NamedNodeMap интерфейс

```
NamedNodeMap myAttributes = myElement.getAttributes();
NamedNodeMap myEntities = myDocument.getEntities();
NamedNodeMap myNotations = myDocument.getNotations();

int getLength();
Node item(int index);
Node getNamedItem(String name);
Node setNamedItem(Node node) throws DOMException; //Node!
Node removeNamedItem(String name) throws DOMException;
```

Node object

Required. The node you want to add/replace  
in the NamedNodeMap collection

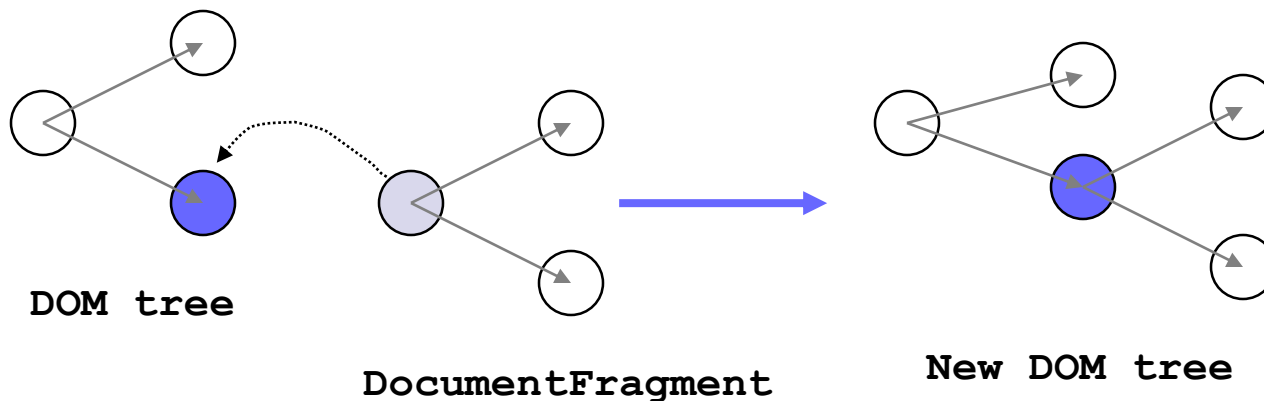
**Tip:** Instead of working with attribute nodes, you  
could use the *element.setAttribute()* method to  
add an attribute with a value to an element.

XML

DOM

# DocumentFragment::Node интерфейс

- Фрагмент от **Document** може да бъде временно съхранен в **DocumentFragment** възел
  - Напр. за 'cut-n-paste'
- Разрушава се, когато се прикачи към друг **Node**



# DOMImplementation интерфейс

- За определяне на нивото на поддръжка в DOM парсър
  - `hasFeature(String feature, String version);`
  - `if (theParser.hasFeature("XML", "1.0") {  
 //XML is supported  
 ...  
}`



# Интерфейс **DOMException**

- NOT\_FOUND\_ERR
  - Опит да се получи референция към възел, който не съществува в контекста
- DOMSTRING\_SIZE\_ERR
  - Опит да се специфицира част от текст с некоректни граници
- HIERARCHY\_REQUEST\_ERR
  - Опит да се вмъкне възел на неподходящо място в йерархията на DOM дървото
- INDEX\_SIZE\_ERR
  - Опит да се достъпи отрицателен индекс или индекс, който е по-голям от допустимата стойност
- NOT\_SUPPORTED\_ERR
  - Опит да се достъпи обект, чийто тип не се поддържа от имплементация

# Разширени интерфейси



## CDATASection

- Добавя CDATA секция
- `<![CDATA []]>`



## DocumentType

- Списък от предефинирани свойства на документа



## Notation

- Декларирана в DTD нотация



## Entity

- Представя възел Entity (специални символи)



## EntityReference

- Представя възел EntityReference (мнемоничен псевдоним на символ)



## ProcessingInstruction

- Добавя инструкции за обработка
- Свойства target и data

# CDATASection::Text интерфейс

- Представя **CDATA** секция (немаркиран текст) – в него се разпознава като край на CDATA секция само разделителя "]]>"
- Атрибутът **DOMString** на възел **Text** съдържа текста на CDATA секция
- Няма добавени методи към **CharacterData**
- Метод-фабрика в **Document** за създаване
  - `CDATASection newCDATA = myDoc.createCDATASection("press <<<ENTER>>>");`

# ProcessingInstruction::Node интерфейс

- Представя декларация за инструкция за обработка
  - Име на възела е *target application name*
  - Стойност на възела е *target application command*

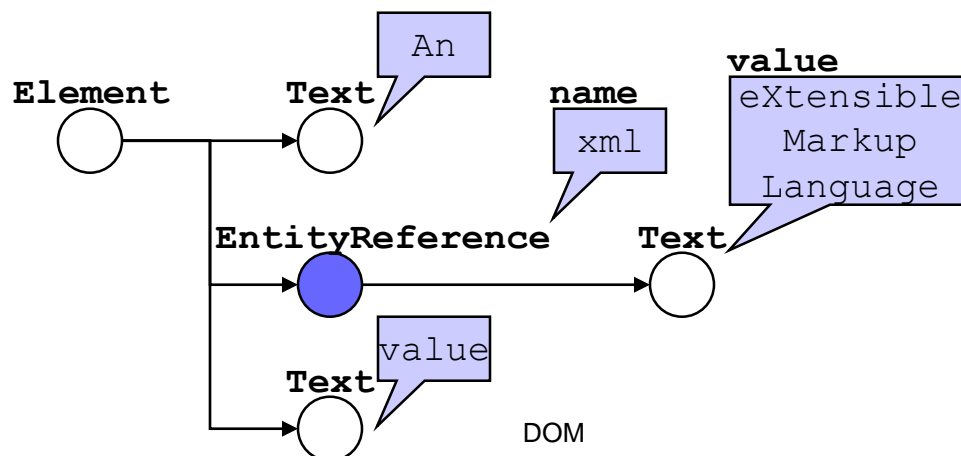
```
//Get the content of the processing instruction
String getData()
//Set the content of the processing instruction
void setData(String data)
//The target of this processing instruction
String getTarget();
```

- Метод-фабрика в **Document** за създаване
  - `ProcessingInstruction newPI =  
myDoc.createProcessingInstruction("ACME",  
"page-break");`

# EntityReference::Node интерфейс

- DOM включва интерфейси за работа с нотации, единици (entities) и референции към тях (entity references)
  - Ако единиците не са били заменени от парсъра със съдържанието им:

```
<!ENTITY xml "eXtensible Markup Language">
<para>An &xml; value</para>
```



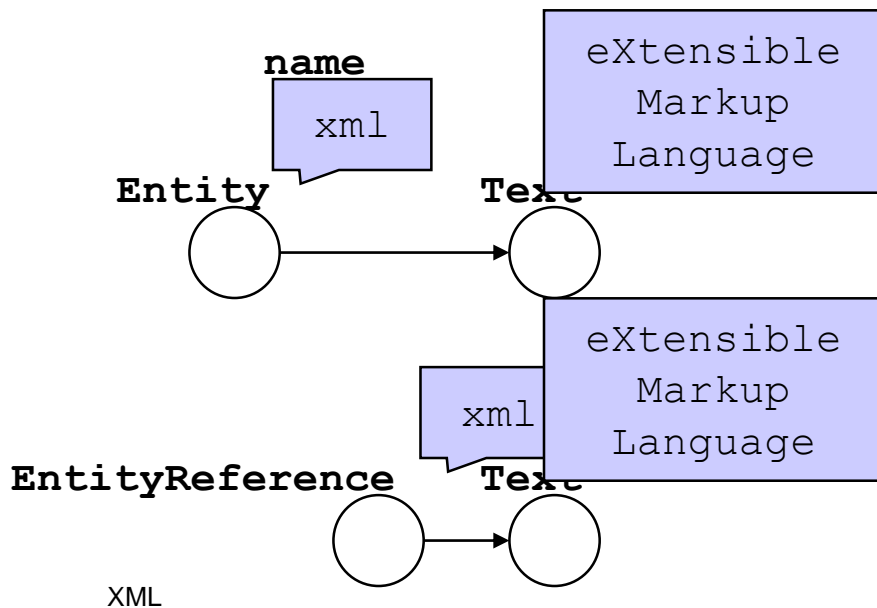
# Entity::Node интерфейс

- Представя единица (parsed или unparsed в XML документ)
  - Парсърът може да замени entity references или да създаде **EntityReference** възли
- Разширява интерфейса **Node** и добавя нови методи
- За non-parsable entities – достъп до името на нотацията:

```
String getPublicId() ;
String getSystemId() ;
String getNotationName() ;
```

# Entity::Node интерфейс (2)

- Едно parsable **Entity** може да има възел дете, който представя стойността за замяна на единицата
- Всички единици на **Document** са достъпни чрез МЕТОДА **getEntities()** на **DocumentType**



```
<!ENTITY MyBoat PUBLIC "BOAT" SYSTEM
"boat.gif" NDATA GIF>
```

```
String publicId = ent.getPublicId() ;
//BOAT
```

```
String systemId = ent.getSystemId() ;
//boat.gif
```

```
String notation =
ent.getNotationName() ; //GIF
DOM
```

# Notation::Node интерфейс

- Всяка декларация на нотация в DTD е представена чрез възел **Notation**
- Към интерфейса **Node** са добавени методите:
  - ```
//Returns content of PUBLIC identifier  
String getPublicId();
```
 - ```
//Returns content of SYSTEM identifier
String getSystemId();
```
- Всичките нотации в даден **Document** са достъпни чрез метода **getNotations()** на обекта **DocumentType**



# DocumentType::Node интерфейс

- Информация за съдържанието на DTD
- DOM 1.0 не позволява редактирането на ТОЗИ възел

//Returns name of document

String                      getName () ;

//Returns general entities declared in DTD

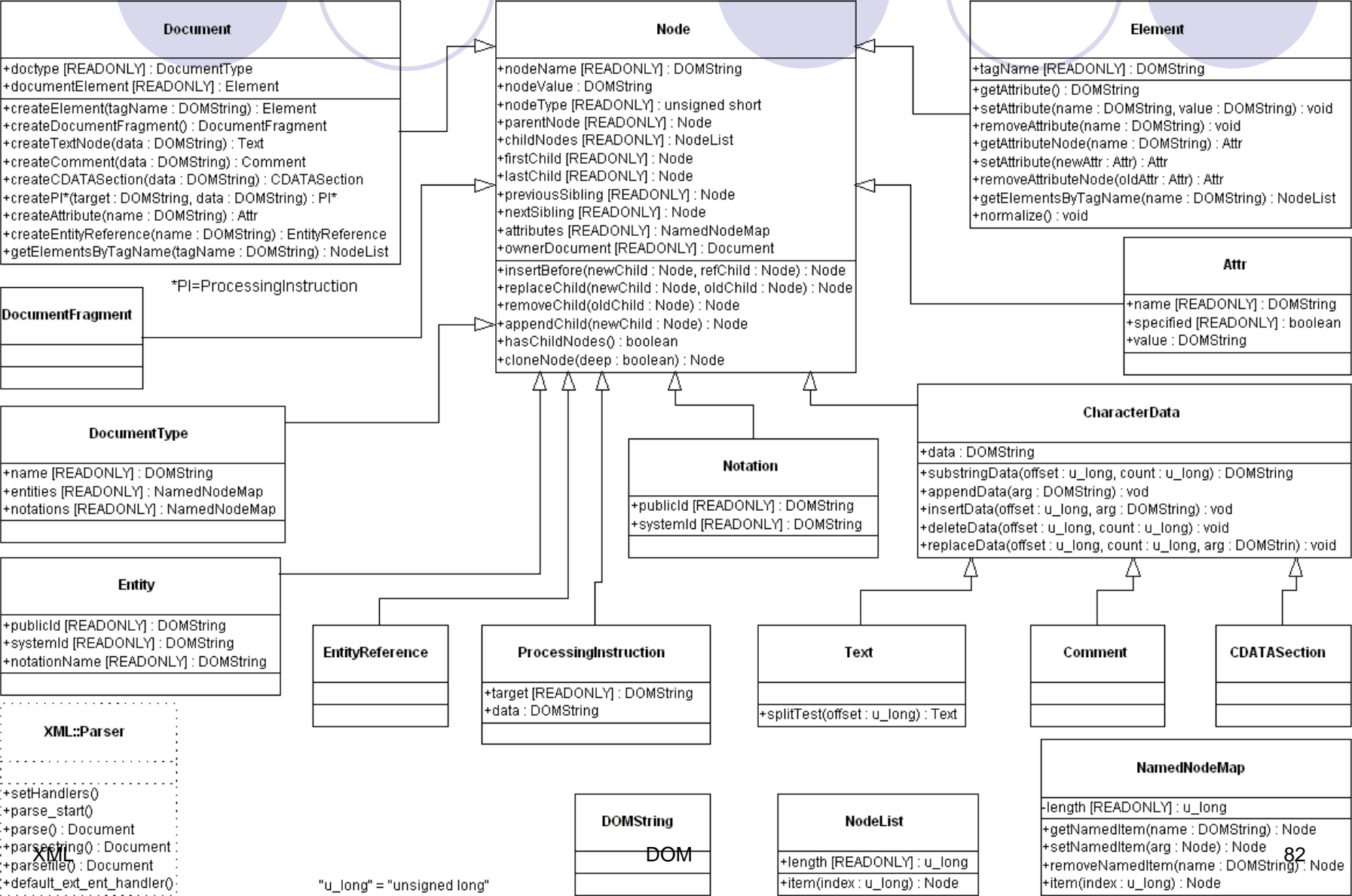
NamedNodeList              getEntities () ;

//Returns notations declared in DTD

NamedNodeList              getNotations () ;

# DOM (Core) Level One

Invoke "getName" to read instance variable *name* when using XML::DOM or XML4J



"u\_long" = "unsigned long"

# Създаване на XML документ from Scratch (DHTML)

```
<script language="JavaScript">
 var oDOM;
 oDOM = new ActiveXObject("MSXML.DOMDocument");
 var oNode, oText;
 oNode = oDOM.createElement("root");
 oText = oDOM.createTextNode("root PCDATA");
 oDOM.appendChild(oNode); oNode.appendChild(oText);

 var oAttr;
 oAttr = oDOM.createAttribute("id"); //set the attribute's value
 oAttr.nodeValue = "123"; //append the attribute to the element
 oNode.attributes.setNamedItem(oAttr);

 alert(oDOM.xml);
</script>
```



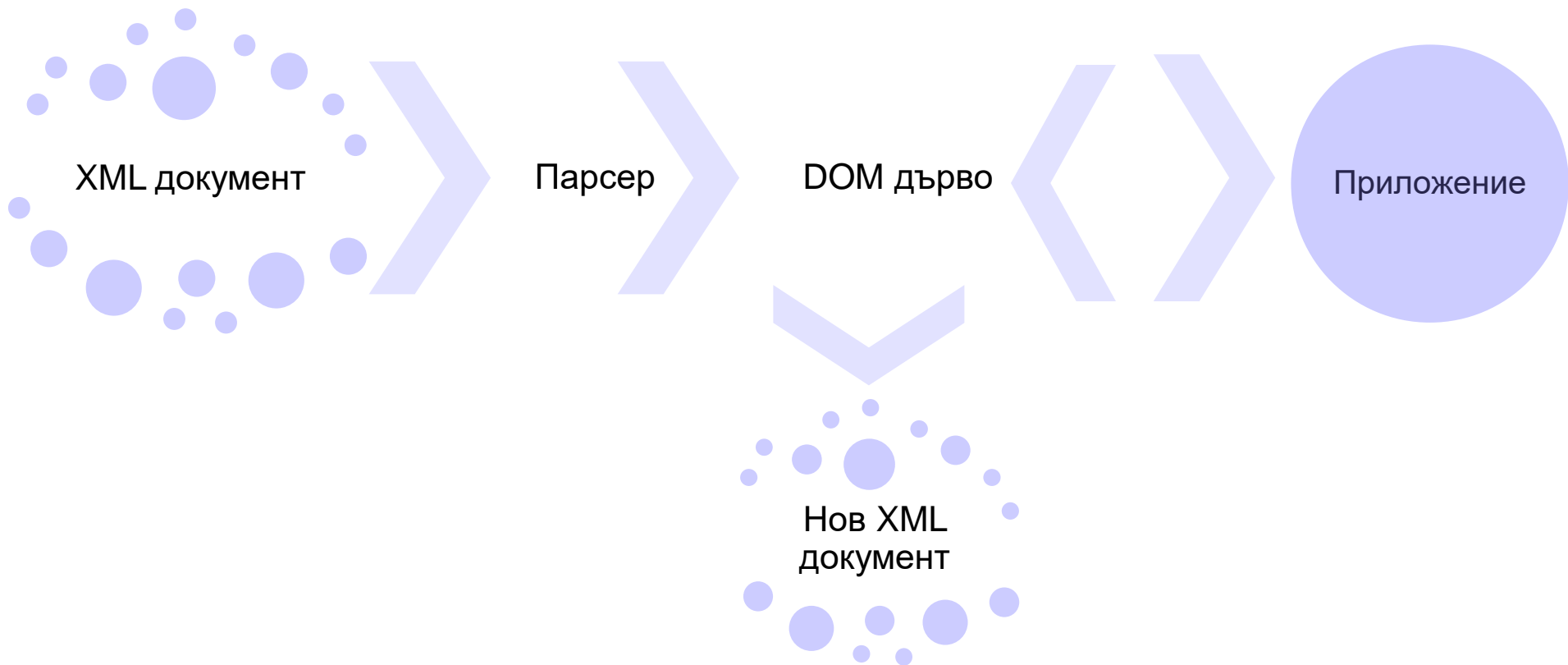
# DOM обекти

- DOM обект  $\Leftrightarrow$  компилиран XML
- Спестяваме време и усилия ако изпращаме и получаваме DOM обекти вместо XML сорс – *използвайте сериализация!*
  - Спестяваме парсването на XML до DOM при изпращача и приемника
  - DOM обектът може да бъде по-голям от XML сорса

# Размер на документа

- Натоварване на паметта
  - DOM изисква представяне на XML документа в паметта
- Осигуряване на интегритет
  - Заклучване на документа при промяна на данните в него
  - Полза: предпазване на потребителите от получаване на некоректни данни или промяна на данни от друг потребител в процес на обработка
- Намаляване на системния товар
  - Използване на фрагменти
  - Забележка: За да бъде получен фрагмент, е необходимо да се зареди пълният документ

# Обработка на документ с DOM

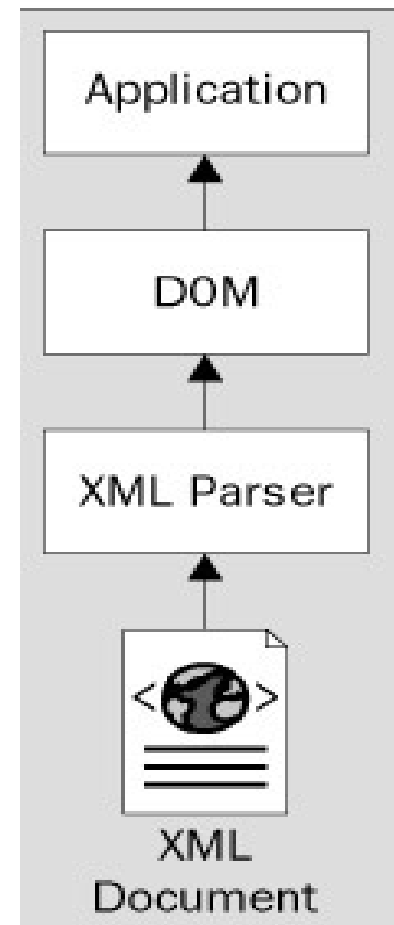


# DOM спрямо XSL

- За сложно сортиране и реструктуриране на документа – използвайте DOM
- При DOM парсване XML документа и после ползваме програмен код за манипулиране на DOM дървото. Този код има пълен достъп до възлите на дървото, без ограниченията на XSL
- XSL процесорът трансформира входния XML документ опосредствено – на база на правила, зададени в друг XML документ

# DOM спрямо SAX

- При големи документи и ако извличаме само някои елементи – по-добре SAX
- При обработка на много елементи и структурни промени в XML документа – по-добре DOM
- При многократен достъп до XML документа – по-добре DOM.





The text is centered and surrounded by five light purple circles. Two circles are positioned above the text, and three are below it. The top-left circle is an outline, while the others are solid.

За домашна работа

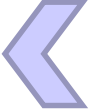
# XML документ

```
<?xml version="1.0"?>
 <SalesData Status="NewVersion">
 <Invoice InvoiceNumber="1"
 TrackingNumber="1"
 OrderDate="01012000"
 ShipDate="07012000"
 ShipMethod="FedEx"
 CustomerIDREF="Customer2">
 <LineItem Quantity="2" Price="5"
 PartIDREF="Part2" />
 </Invoice>
 </SalesData>
```

```
</Invoice>
 <Customer ID="Customer2"
 firstName="Bob"
 lastName="Smith"
 Address="2AnyStreet"
 City="Anytown" State="AS"
 PostalCode="ANYCODE" />
 <Part PartID="Part2" PartNumber="13"
 Name="Winkle" Color="Red"
 Size="10" />
</SalesData>
```

# Създаване на HTML документ и инстанция на Microsoft MSXML парсер

```
<HTML>
 <HEAD>
 <TITLE>DOM Demo</TITLE>
 <SCRIPT language="JavaScript">
 ...
 </SCRIPT>
 </HEAD>
 <BODY>
 <P>We have retrieved a lot<P>
 </BODY>
</HTML>
```



```
var objDOM;
objDOM = new ActiveXObject("MSXML2.DOMDocument");
objDOM.async = false;
objDOM.load("salesData.xml");
```

# Извеждане на кореновия елемент

```
//Get to the root element
document.write("The root element:
");
varSalesData = objDOM.documentElement;
alert(varSalesData.tagName);
document.write(varSalesData.tagName);
```

```
<?xml version="1.0"?>
 <SalesData Status="NewVersion">
 <Invoice InvoiceNumber="1"
 TrackingNumber="1"
 OrderDate="01012000"
 ShipDate="07012000"
 ShipMethod="FedEx"
 CustomerIDREF="Customer2">
 <LineItem Quantity="2" Price="5"
 PartIDREF="Part2" />
 </Invoice>
 </SalesData>
```

# Достъп до втория наследник на кореновия елемент

```
//Find the Customer elements and select the first one
document.write
("<P>The name of the first Customer Element: ");
varElemCust1 =

varSalesData.getElementsByTagName("Customer").item(0);
alert(varElemCust1.xml);
document.write(varElemCust1.tagName);
```

```
<Customer
 ID="Customer2"
 firstName="Bob"
 lastName="Smith"
 Address="2AnyStreet"
 City="Anytown" State="AS"
 PostalCode="ANYCODE" />
```

# Достъп до ID атрибута на елемента Customer

```
//Find the Customer ID Attribute
document.write
("<P>The Customer ID attribute is: ");
varAttrCustID = varElemCust1.getAttribute("ID");
alert(varAttrCustID);
document.write(varAttrCustID);
```

```
<Customer
 ID="Customer2"
 firstName="Bob"
 lastName="Smith"
 Address="2AnyStreet"
 City="Anytown" State="AS"
 PostalCode="ANYCODE" />
```

# Достъп до атрибутите firstName и lastName на елемента customer

```
//Find the next attribute of Name
document.write("<P>The customer's name is: ");
varAttrFirstName =
varElemCust1.getAttribute("firstName");
alert(varAttrFirstName);
document.write(varAttrFirstName);
varAttrLastName = varElemCust1.getAttribute("lastName");
alert(varAttrLastName);
document.write(varAttrLastName);
```

```
<Customer
 ID="Customer2"
 firstName="Bob"
 lastName="Smith"
 Address="2AnyStreet"
 City="Anytown" State="AS"
 PostalCode="ANYCODE" />
```

# Достъп до атрибутите ADDRESS и City на елемента customer

```
//Now let's write out the address
document.write("<P>Their address is: ");
varAttrAddr = varElemCust1.getAttribute("Address");
alert(varAttrAddr);
document.write(varAttrAddr);
//Find the next attribute of City
varAttrCity = varElemCust1.getAttribute("City");
alert(varAttrCity);
document.write(varAttrCity);
```

```
<Customer
 ID="Customer2"
 firstName="Bob"
 lastName="Smith"
 Address="2AnyStreet"
 City="Anytown" State="AS"
 PostalCode="ANYCODE" />
```



# Добавяне на елемент в документа

```
//Find the Customer elements and select the first one
varElemCust1 = varSalesData.getElementsByTagName("Customer").item(0);
<!-- adding an element -->
document.write("<HR><H1>Updates appear in alert boxes:</H1>");
//create a new element
varNewElem = objDOM.createElement("MonthlySalesData");
//append the element
varNewElem = varSalesData.insertBefore(varNewElem, varElemCust1);
```

# Добавяне на съдържание в елемент и създаване на атрибут

```
//create a new text-type node and append it
newText = objDOM.createTextNode("Can you see that we have created a new element?");
varNewElem.appendChild(newText);
alert(objDOM.xml);
<!-- adding an attribute -->
//create a new attribute and give it a value
varElemCust1.setAttribute("telephoneNo", "3591765524");
```

# Добавяне на информация от друго DOM дърво

```
<?xml version="1.0"?>
 <SalesData Status="NewVersion">
 <Invoice InvoiceNumber="1"
 TrackingNumber="1"
 OrderDate="01012000"
 ShipDate="07012000"
 ShipMethod="FedEx"
 CustomerIDREF="Customer2">
 <LineItem Quantity="2" Price="5"
 PartIDREF="Part2" />
 </Invoice>
```

```
 <Customer ID="Customer1"
 firstName="Tom"
 lastName="Boswell"
 Address="39BrownhillCrescent"
 City="Anothertown" State="IN"
 PostalCode="OTHERCODE" />
 <Part PartID="Part2" PartNumber="13"
 Name="Winkle" Color="Red"
 Size="10" />
 </SalesData>
```

# Получаване на достъп до кореновия елемент на първото DOM дърво

```
<SCRIPT language="JavaScript">
 var objDOM;
 objDOM = new ActiveXObject("MSXML2.DOMDocument");
 objDOM.async = true;
 objDOM.load("salesData.xml");
 //Get to the root element
 varSalesData = objDOM.documentElement;
 ...
</SCRIPT>
```

```
<?xml version="1.0"?>
 <SalesData Status="NewVersion">
 <Invoice InvoiceNumber="1"
 TrackingNumber="1"
 OrderDate="01012000"
 ShipDate="07012000"
 ShipMethod="FedEx"
 CustomerIDREF="Customer2">
 <LineItem Quantity="2" Price="5"
 PartIDREF="Part2" />
 </Invoice>
 </SalesData>
```

# Получаване на достъп до елемент Customer във второто DOM дърво

```
//second instance of the DOM
var objSecondDOM;
objSecondDOM =
 new
 ActiveXObject("MSXML2.DOMDocument");
objSecondDOM.async = true;
objSecondDOM.load("salesData2.xml");
//Get to the root element
varSalesDataB =
objSecondDOM.documentElement;
varImportCust1 =

varSalesDataB.getElementsByTagName("C
ustomer").item(0);
```

```
<Customer ID="Customer1"
 firstName="Tom"
 lastName="Boswell"
 Address="39BrownhillCrescent"
 City="Anothertown"
 State="IN"
 PostalCode="OTHERCODE" />
```

# Клониране на елемента Customer и добавяне на новия елемент в първото DOM дърво

```
<!-- adding an element -->
document.write("<HR><H1>Updates appear in alert
boxes:</H1>");
//clone the node from the second DOM
varClone = varImportCust1.cloneNode(true);
//append node to the first DOM
varSalesData.appendChild(varClone);
```