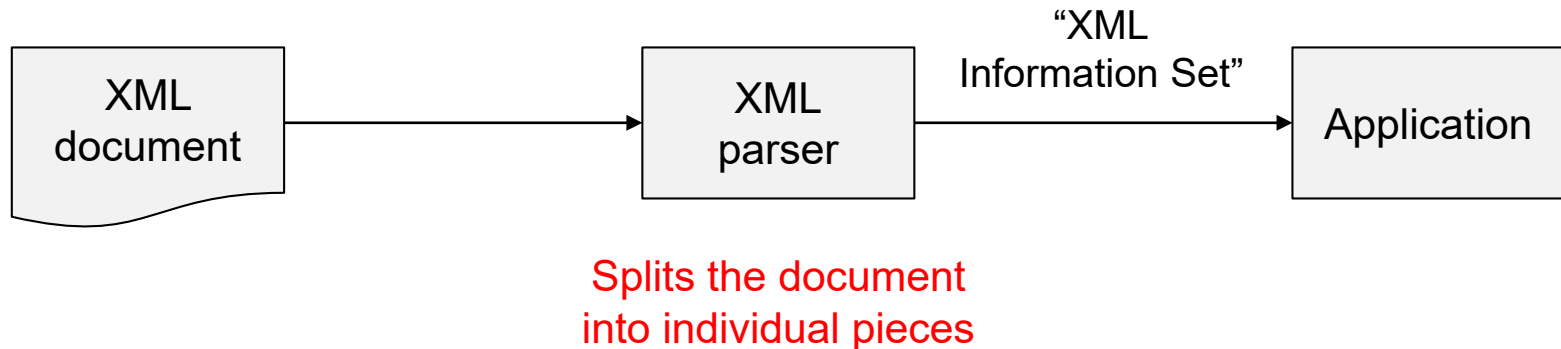


Semi-structured Data

Document Object Model (DOM)

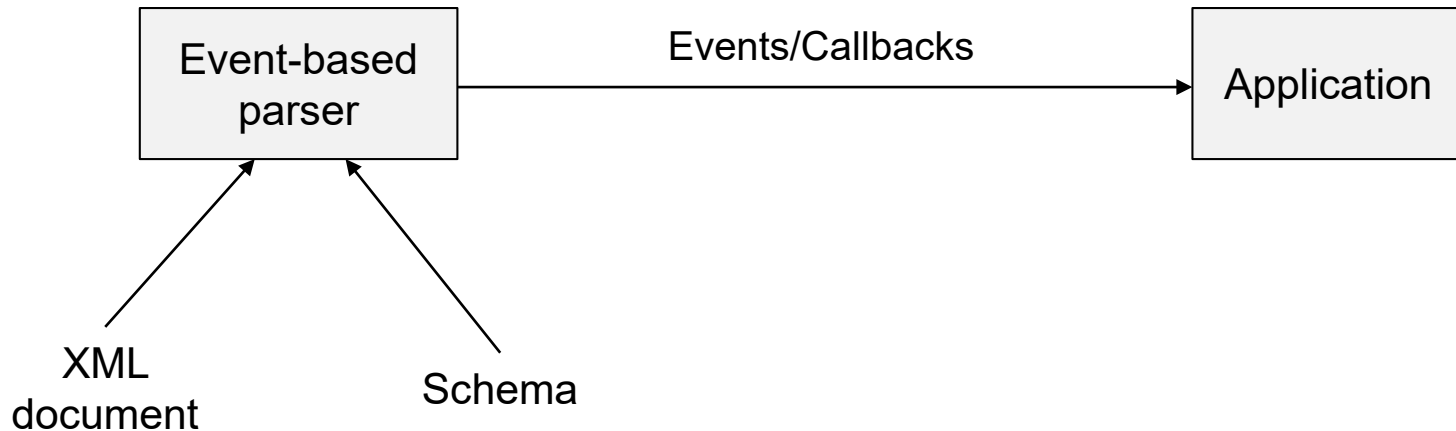
How XML Works

- Strict rules regarding the syntax of XML documents - allows for the development of **XML parsers** that can read documents
- Applications that need to understand an XML document will use a parser



Event-Based Parsers

- Report **parsing events**, such as the start and end of elements, directly to the application
- The application implements **handlers** to deal with the different events



Event-Based Parsers

parse

<element attr="attr-value">
 ...text-1...
 <subelement>...text-2...</subelement>
</element>

Events/Callbacks

start document

start element: "element"

attribute name="attr" value="attr-value"

characters: "...text-1..."

start element: "subelement"

characters: "...text-2..."

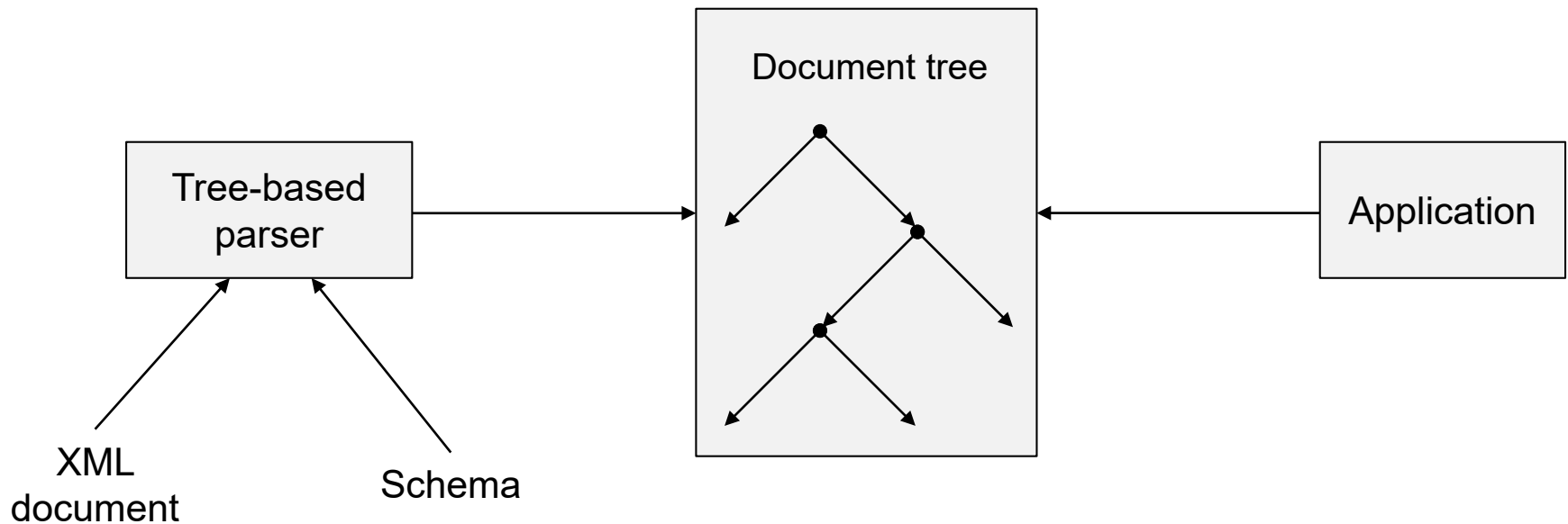
end element: "subelement"

end element: "element"

end document

Tree-Based Parsers

- Map an XML document into an **internal tree structure** stored in main memory
- The application **navigates** that tree



Tree-Based Parsers

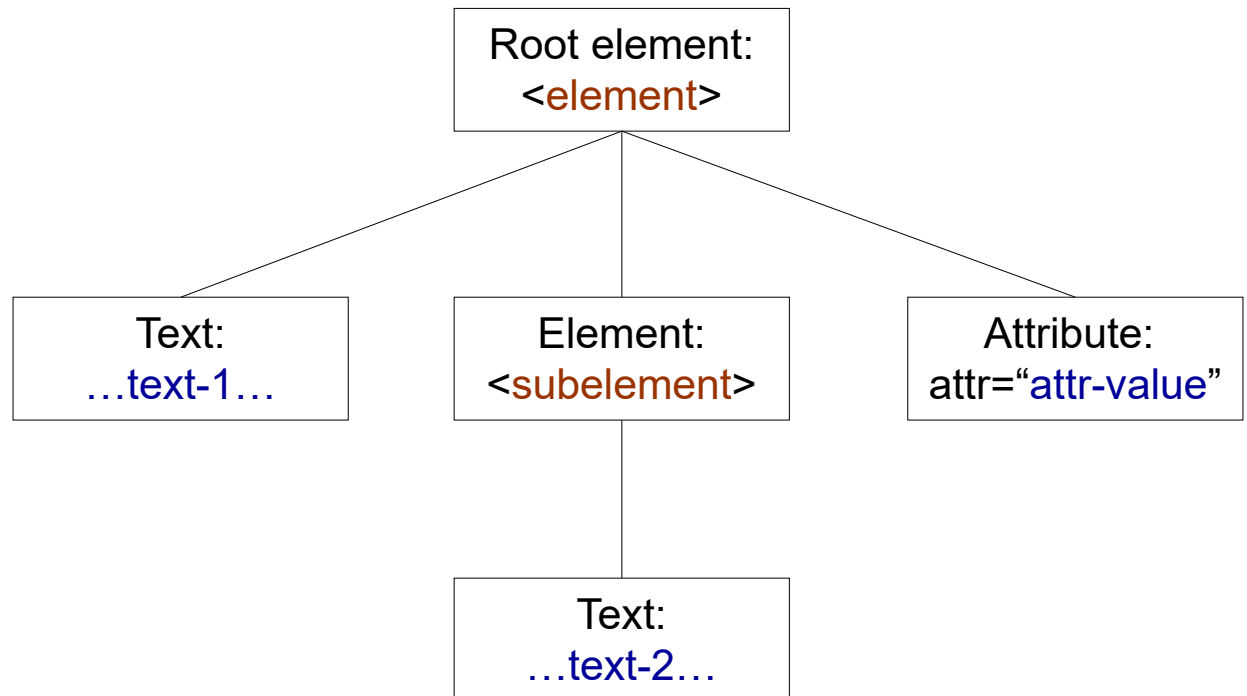
`<element attr="attr-value">`

`...text-1...`

`<subelement>...text-2...</subelement>`

`</element>`

Document Tree



Event-Based vs. Tree-Based Parsers

Event-based	Tree-based
<ul style="list-style-type: none">• Sequential access• Fast• Constant memory	<ul style="list-style-type: none">• Random access• Slow• Proportional to the document size
+	
<ul style="list-style-type: none">• Large documents• Lack of data structure	<ul style="list-style-type: none">• Small documents• Ready-made data structure

Standards for XML Parsers

- **SAX** - Simple API for XML (event-based)
 - “De facto” standard
- **DOM** - Document Object Model (tree-based)
 - W3C standard



... APIs to read and interpret XML documents

... we have seen SAX last week and focus on DOM today

Outline

- DOM (Nodes, Node-tree)
- Load an XML Document
- The Node Interface
- Subinterfaces of Node
- Reading a Document
- Creating a Document

DOM - Document Object Model

- A **tree-based** API for reading and manipulating documents like XML and HTML
- A W3C standard
- The XML DOM defines the objects and properties of all XML elements, and the methods to access them
- The XML DOM is a standard for how to get, change, add or delete XML elements

DOM Nodes

Everything in an XML document is a **node**

The document is a **document node**

Every element is an **element node**

Text in an element is a **text node**

Every attribute is an **attribute node**

A comment is a **comment node**

ATTENTION: Element nodes do not contain text

DOM Node Tree

- An XML document is seen as a tree-structure - **node-tree**
- All nodes can be accessed through the node-tree
- Nodes can be modified/deleted, and new elements can be created

DOM Node Tree: Example

```
<?xml version="1.0"?>
<courses>
  <course semester="Summer">
    <title> Semi-structured Data (SSD) </title>
    <day> Thursday </day>
    <time> 09:15 </time>
    <location> HS8 </location>
  </course>
</courses>
```

DOM Node Tree: Example

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

```
<courses>
```

```
  <course semester="Summer">
```

```
    <title> Semi-structured Data (SSD) </title>
```

```
    <day> Thursday </day>
```

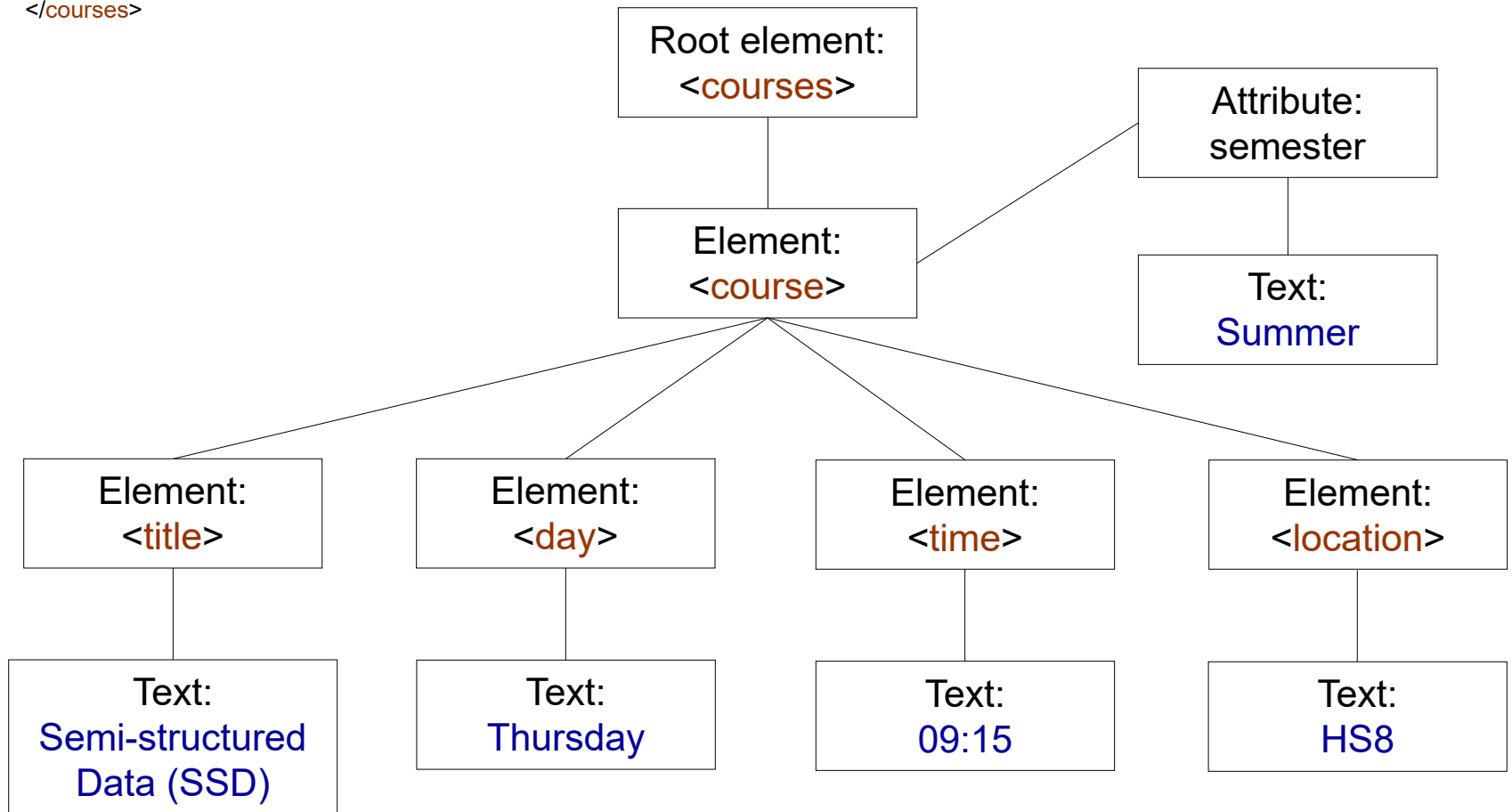
```
    <time> 09:15 </time>
```

```
    <location> HS8 </location>
```

```
  </course>
```

```
</courses>
```

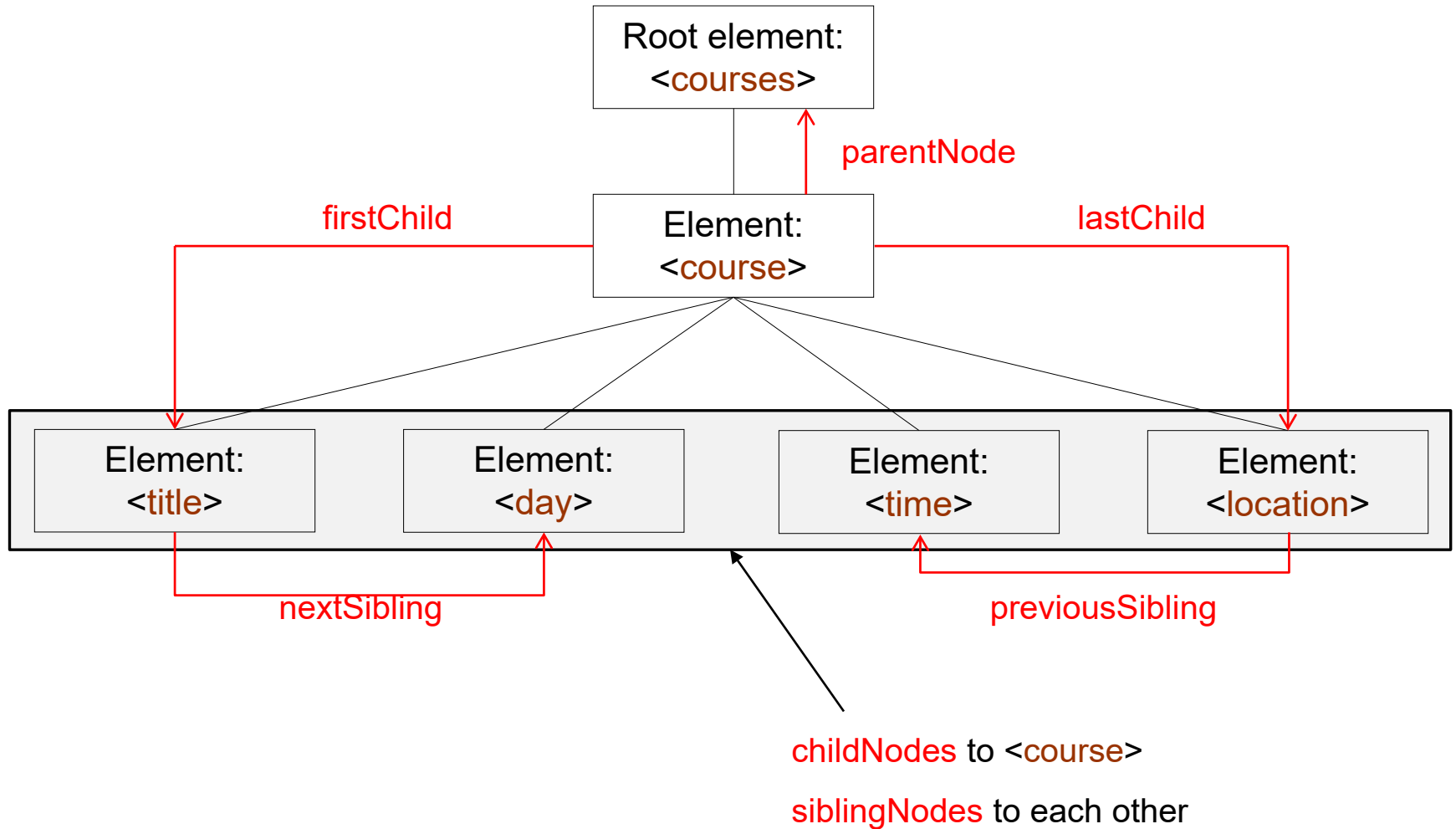
DOM Node Tree



Relationships Among Nodes

- The terms **parent**, **child** and **sibling** are describing the relationships among nodes
- In a node-tree:
 - The top node is the root
 - Every node has exactly one parent (except the root)
 - A node can have an unbounded number of children
 - A leaf node has no children
 - Siblings have the same parent

Relationships Among Nodes



XML DOM Parser

- The parser converts the document into an XML DOM object that can be accessed with Java
- XML DOM contains methods to traverse node-tree, access, insert and delete nodes

Load an XML Document into a DOM Object

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //factory instantiation
        //factory API that enables applications to obtain a parser that
        //produces DOM object trees from XML documents
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        //validation and namespaces
        factory.setValidating(true);
        factory.setNamespaceAware(true);
        //parser instantiation
        //API to obtain DOM document instances from XML documents
        DocumentBuilder builder = factory.newDocumentBuilder();
        //install ErrorHandler
        builder.setErrorHandler(new MyErrorHandler());
        //parsing instantiation
        Document coursedoc = builder.parse(args[0]);
    }
} //end of Course class
```

Class MyErrorHandler

```
import org.xml.sax.*;
```

```
public class MyErrorHandler implements ErrorHandler {  
    public void fatalError(SAXParseException ex) throws SAXException {  
        printError("FATAL ERROR", ex)  
    }  
  
    public void error(SAXParseException ex) throws SAXException {  
        printError("ERROR", ex)  
    }  
  
    public void warning(SAXParseException ex) throws SAXException {  
        printError("WARNING", ex)  
    }  
  
    private void printError(String err, SAXParseException ex) {  
        System.out.printf("%s at %3d, %3d: %s \n", err, ex.getLineNumber(), ex.getColumnNumber(),  
                           ex.getMessage());  
    }  
} // end of MyErrorHandler class
```

Load an XML Document into a DOM Object

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //factory instantiation
        //validation and namespaces
        //parser instantiation
        //install ErrorHandler
        //parsing instantiation
    }
} //end of Course class
```

ATTENTION: We set up the document builder, and also error handling is in place. However, Course does not do anything yet.

Up to Now

- **DOM (Nodes, Node-tree)**
- **Load an XML Document**
- The Node Interface
- Subinterfaces of Node
- Reading a Document
- Creating a Document

The Node Interface

- The **primary datatype** of the entire DOM
- It represents a single node in the node-tree
- It is the base interface for all the other (more specific) nodes (Document, Element, Attribute, etc.)

Subinterfaces of Node

- There is a separate interface for each node type that might occur in an XML document
- All node types inherit from class Node
- Some important **subinterfaces of Node**:
 - Document - the document
 - Element - an element
 - Attr - an attribute of an element
 - Text - textual content

A Simple Example

- Visit all child nodes of a node

```
private void visitNode(Node node) {  
    //iterate over all children  
    for (int i = 0; i < node.getChildNodes().getLength(); i++) {  
        //recursively visit all nodes  
        visitNode(node.getChildNodes().item(i));  
    }  
}
```

- Go through all the nodes of courses.xml

```
visitNode(coursedoc.getDocumentElement());
```



the root node of the node-tree representing courses.xml

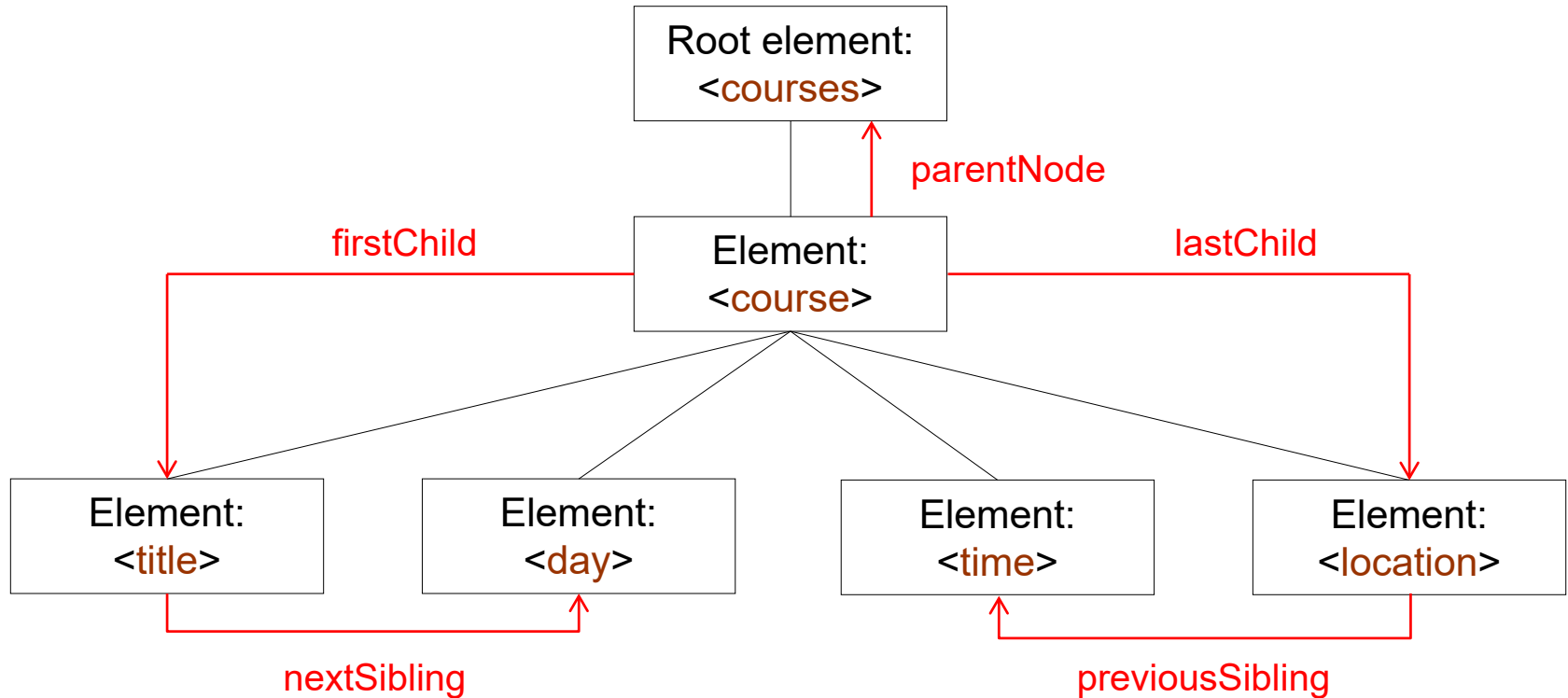
Node Methods

- `public String getNodeName()`
- `public String getNodeValue()`
- `public String getTextContent()`
- `public short getNodeType()`
- `public String getNamespaceURI()`
- `public String getPrefix()`
- `public String getLocalName()`

... more details for these methods
can be found in the DOM-methods slides

<http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Node.html>

Recall the Relationships Among Nodes



Node Methods

- `public Node getParentNode()`
- `public boolean hasChildNodes()`
- `public NodeList getChildNodes()`
- `public Node getFirstChild()`
- `public Node getLastChild()`

abstraction of an ordered collection of nodes

- `int getLength()` - number of nodes in the list
- `Node item(int i)` - i-th node in the list; null if i is not a valid index

- `public Node getPreviousSibling()`
- `public Node getNextSibling()`
- `public boolean hasAttributes()`
- `public NamedNodeMap getAttributes()`

collection of nodes that can be accessed by name

- `int getLength()` - number of nodes in the map
- `Node getItem(String name)` - retrieves a node by name; null if it does not identify any node in the map
- `Node item(int i)` - i-th node in the map; null if i is not a valid index

Node Methods

- `public Node getParentNode()`
- `public boolean hasChildNodes()`
- `public NodeList getChildNodes()`
- `public Node getFirstChild()`
- `public Node getLastChild()`
- `public Node getPreviousSibling()`
- `public Node getNextSibling()`
- `public boolean hasAttributes()`
- `public NamedNodeMap getAttributes()`

- If a node does not exist, then we get null
- A NodeList may be empty (no child nodes)
- `getAttributes()` from elements; otherwise, null

Node Methods

- `public Node insertBefore(Node newChild, Node refChild)`
- `public Node replaceChild(Node newChild, Node oldChild)`
- `public Node removeChild(Node oldChild)`
- `public Node appendChild(Node newChild)`
- `public Node cloneNode(boolean deep)`

... more details for these methods
can be found in the DOM-methods slides

<http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Node.html>

Up to Now

- **DOM (Nodes, Node-tree)**
- **Load an XML Document**
- **The Node Interface**
- Subinterfaces of Node
- Reading a Document
- Creating a Document

Subinterfaces of Node

- There is a separate interface for each node type that might occur in an XML document
- All node types inherit from class Node
- Some important **subinterfaces of Node**:
 - Document - the document
 - Element - an element
 - Attr - an attribute of an element
 - Text - textual content
 - ...
- Subinterfaces provide **useful additional methods**

Document Interface

- It provides **methods to create new nodes**:
 - `Attr createAttribute(String name)`
 - `Element createElement(String tagName)`
 - `Text createTextNode(String data)`

... more details for these methods
can be found in the DOM-methods slides

<http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Document.html>

Element Interface

- `NodeList getElementsByTagName(String name)`
- `boolean hasAttribute(String name)`
- `String getAttribute(String name)`
- `void setAttribute(String name, String value)`
- `void removeAttribute(String name)`
- `Attr getAttributeNode(String name)`
- `Attr setAttributeNode(Attr newAttr)`
- `Attr removeAttributeNode(Attr oldAttr)`

... more details for these methods
can be found in the DOM-methods slides

<http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Element.html>

Attribute Interface

- `String getName()`
- `String getValue()`
- `Element getOwnerElement()`

... more details for these methods
can be found in the DOM-methods slides

<http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Attr.html>

Up to Now

- **DOM (Nodes, Node-tree)**
- **Load an XML Document**
- **The Node Interface**
- **Subinterfaces of Node**
- Reading a Document
- Creating a Document

Example: Reading the Whole Document

courses.xml

```
<?xml version="1.0"?>
<courses>
  <course semester="Summer">
    <title> Semi-structured Data (SSD) </title>
    <day> Thursday </day>
    <time> 09:15 </time>
    <location> HS8 </location>
  </course>
</courses>
```

Expected Result

```
courses:
course: semester="Summer"
title: "Semi-structured Data (SSD)"
day: "Thursday"
time: "09:15"
location: "HS8"
```

Example: Reading the Whole Document

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //preliminary code - already discussed
        Document coursedoc = builder.parse(args[0]);

        //call visit node starting from the root node
        visitNode(coursedoc.getDocumentElement());
    }

    //the recursive method visitNode
    private static void visitNode(Node node) {
        ...
    }
} //end of Course class
```

Example: Reading the Whole Document

```
private static void visitNode(Node node) {
    //element nodes
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        System.out.print("\n" + node.getNodeName() + ": ");
        NamedNodeMap attributes = node.getAttributes();
        if (attributes != null) {
            for (int i = 0; i < attributes.getLength(); i++) {
                System.out.print(attributes.item(i) + " ");
            }
        }
    }
    //text nodes
    if (node.getNodeType() == Node.TEXT_NODE && !node.getTextContent().trim().isEmpty()) {
        System.out.print("\"" + node.getTextContent().trim() + "\"");
    }
    // visit child nodes
    NodeList nodelist = node.getChildNodes();
    for (int i = 0; i < nodelist.getLength(); i++) {
        visitNode(nodelist.item(i));
    }
} //end of visitNode
```

Example: Create New Documents

Create the courses.xml document

```
<courses>
  <course semester="Summer">
    <title> Semi-structured Data (SSD) </title>
    <day> Thursday </day>
    <time> 09:15 </time>
    <location> HS8 </location>
  </course>
</courses>
```

1. Create a new document
2. Create all the necessary elements
3. Append the children in a bottom-up-order

Example: Create New Documents

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //preliminary code - already discussed
        //create a new document
        Document coursedoc = builder.newDocument();

        //create all the necessary elements
        Element courses = coursedoc.createElement("courses");
        Element course = coursedoc.createElement("course");
        course.setAttribute("semester", "Summer");
        Element title = coursedoc.createElement("title");
        title.setTextContent("Semi-structured Data (SSD)");
        //similarly for day, time and location elements

    }
} //end of Course class
```


Example: Create New Documents

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Course {
    public static void main(String[] args) throws Exception {
        //preliminary code - already discussed
        //create a new document
        Document coursedoc = builder.newDocument();

        //create all the necessary elements
        ...

        //append the children in a bottom-up-order
        course.appendChild(title); course.appendChild(day);
        course.appendChild(time); course.appendChild(location);
        courses.appendChild(course);
        coursedoc.appendChild(courses);
    }
} //end of Course class
```

... but, we would like to have the document in a file

Example: Store (New) Documents

```
import java.io.File;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;

public class Course {
    public static void main(String[] args) throws Exception {

    }
} //end of Course class
```

Example: Store (New) Documents



```
public class Course {  
    public static void main(String[] args) throws Exception {  
        //preliminary code - already discussed  
        //create a new document  
  
        //write the document into a file  
        //factory instantiation  
        TransformerFactory tfactory = TransformerFactory.newInstance();  
        //transformer instantiation  
        Transformer transformer = tfactory.newTransformer();  
        //create a new input XML source  
        DOMSource source = new DOMSource(coursedoc);  
        //construct a stream result  
        StreamResult result = new StreamResult(new File("courses.xml"));  
        //actual transformation  
        transformer.transform(source, result);  
        System.out.println("File saved!");  
    }  
} //end of Course class
```

... but, we might want to make sure that the document is valid w.r.t. a given schema

Example: Validate (New) Documents

```
public class Course {  
    public static void main(String[] args) throws Exception {  
        //preliminary code - already discussed  
  
        // validate the new document against a schema  
        //load XML Schema  
        File schemaFile = new File ("schema.xsd");  
        SchemaFactory factory= SchemaFactory(  
            XMLConstants.W3C_XML_SCHEMA_NS_URI);  
        Schema schema = schemaFactory.newSchema(schemaFile);  
        // Create Validator and validate  
        Validator validator = schema.newValidator();  
        try{  
            validator.validate(new DOMSource(coursedoc))  
        } catch (SAXException ex) {  
            exit("Created document not valid:" + ex.getMessage());  
        }  
        //write the document into a file  
    }  
} //end of Course class
```

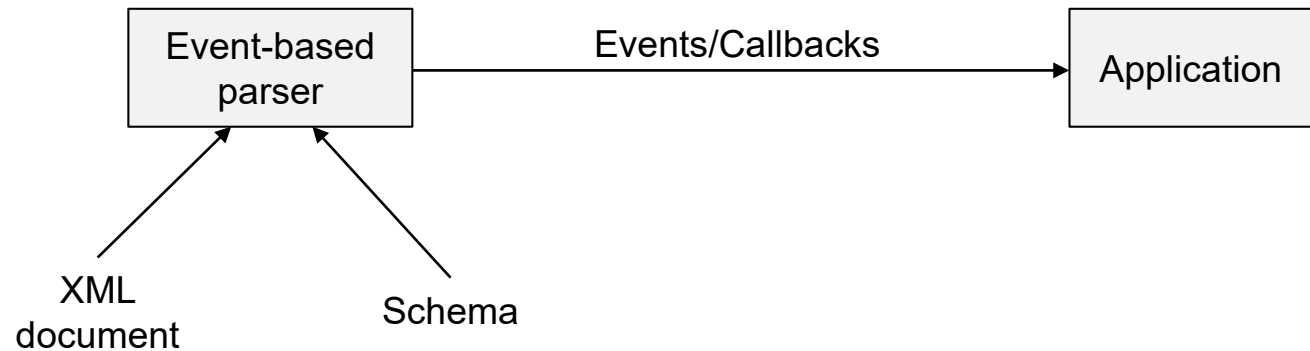
Standards for XML Parsers

- **SAX** - Simple API for XML (event-based)
 - “De facto” standard 
- **DOM** - Document Object Model (tree-based)
 - W3C standard 

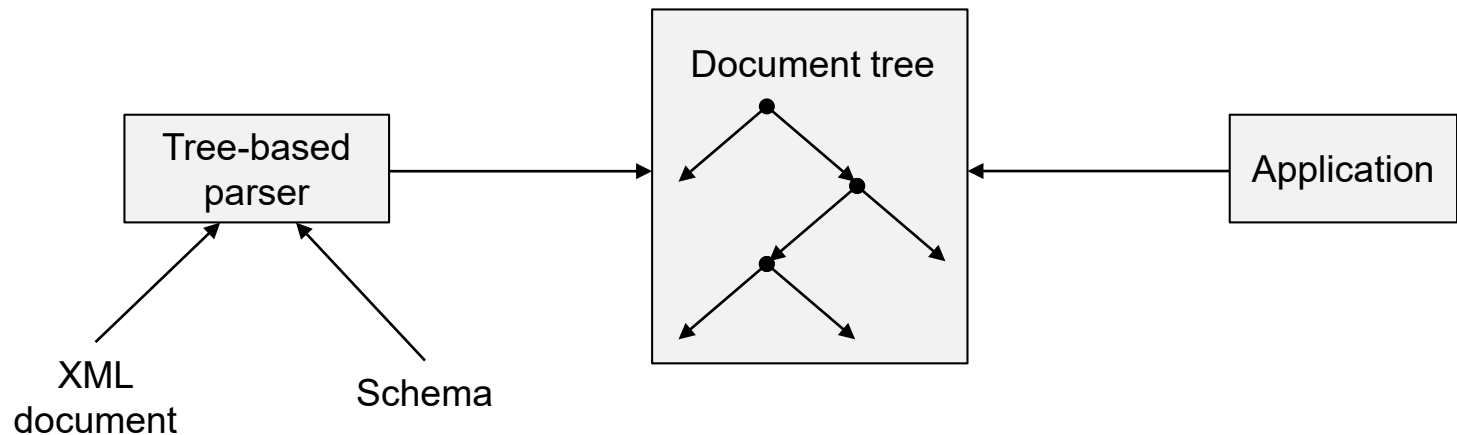
... APIs to read and interpret XML documents

XML Parsers

- Event-based parses



- Tree-based parsers



Comparison of Parsers

Event-based	Tree-based
<ul style="list-style-type: none">• Sequential access• Fast• Constant memory - does not depend on the document	<ul style="list-style-type: none">• Random access• Slow• Proportional to the size of the document
+	
<ul style="list-style-type: none">• Large documents• Lack of data structure	<ul style="list-style-type: none">• Small documents• Ready-made data structure

Semi-structured Data

9 - Document Object Model (DOM) Methods Overview

Node Methods

```
public String getNodeName()
```

- The **name of the node**, depending on its type
 - Document - “#document”
 - Element - Element.tagName
 - Attr - Attr.name
 - Text - “#text”

Node Methods

`public String getNodeValue() throws DOMException`

- The **value of the node**, depending on its type
 - Document - null
 - Element - null
 - Attr - Attr.value
 - Text - the content of the text node

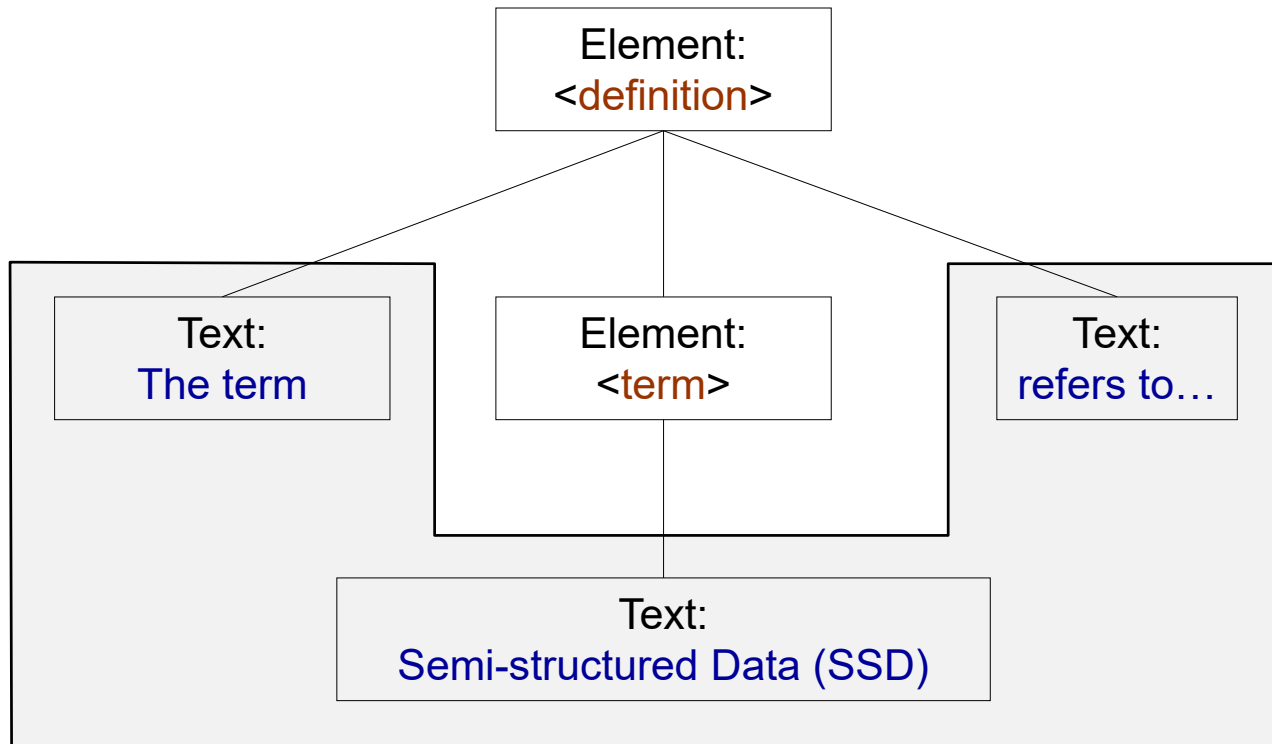
Node Methods

`public String getTextContent() throws DOMException`

- The **text content of the node**, depending on its type
 - Document - null
 - Element
 - Attr
 - Text - the content of the text node
- concatenation of the text content of every child node

Node Methods

public String getTextContent() throws DOMException



The term Semi-structured Data (SSD) refers to...

Node Methods

```
public short getNodeTypes()
```

- A **code** representing the type of the underlying object
 - Document - 9 (**DOCUMENT_NODE**)
 - Element - 1 (**ELEMENT_NODE**)
 - Attr - 2 (**ATTRIBUTE_NODE**)
 - Text - 3 (**TEXT_NODE**)

Node Methods

```
public String getNamespaceURI()
```

the namespace URI of the node, or null if it is undefined

```
public String getPrefix()
```

the namespace prefix of the node, or null if it is undefined

```
public String getLocalName()
```

the local part of the qualified name of the node

Node Methods

- `public Node getParentNode()`
- `public boolean hasChildNodes()`
- `public NodeList getChildNodes()`
- `public Node getFirstChild()`
- `public Node getLastChild()`

abstraction of an ordered collection of nodes

- `int getLength()` - number of nodes in the list
- `Node item(int i)` - i-th node in the list; null if i is not a valid index

- If a node does not exist, then we get null
- A NodeList may be empty (no child nodes)
- `getAttributes()` from elements; otherwise, null

- `public Node getPreviousSibling()`
- `public Node getNextSibling()`
- `public boolean hasAttributes()`
- `public NamedNodeMap getAttributes()`

collection of nodes that can be accessed by name

- `int getLength()` - number of nodes in the map
- `Node getNamedItem(String name)` - retrieves a node by name; null if it does not identify any node in the map
- `Node item(int i)` - i-th node in the map; null if i is not a valid index

Node Methods

```
public Node insertBefore(Node newChild, Node refChild)  
    throws DOMException
```

- Inserts the node **newChild** before the existing node **refChild**, and returns the inserted node
- If **refChild = null**, then newChild is inserted at the end of the list of children
- If newChild is already in the tree, it is first removed

Node Methods

```
public Node replaceChild(Node newChild, Node oldChild)  
    throws DOMException
```

- Replaces the child node `oldChild` with `newChild` in the list of children, and returns the old child
- If `newChild` is already in the tree, it is first removed

Node Methods

`public Node removeChild(Node oldChild) throws DOMException`

- **Removes the child node `oldChild`** from the list of children, and returns it

`public Node appendChild(Node newChild) throws DOMException`

- **Adds the node `newChild`** to the end of the list of children, and returns it
- If `newChild` is already in the tree, it is first removed

Node Methods

```
public Node cloneNode(boolean deep)
```

- **Returns a duplicate of the node** - a generic copy constructor for nodes
- If **deep = true**, recursively clones the subtree under the specified name
- If **deep = false**, clones only the node itself (and its attributes, in case of an element)

see <http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Node.html>

Document Interface

- It provides **methods to create new nodes**:
 - **Attr createAttribute(String name)** throws **DOMException**
creates an attribute of the given name; its value is the empty string
 - **Element createElement(String tagName)** throws **DOMException**
creates an element of the given name
 - **Text createTextNode(String data)**
creates a text node given the specified string

see <http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Document.html>

Element Interface

- **NodeList getElementsByTagName(String name)**
returns a node list of all descendant elements with the specified tag name, in document order
- **boolean hasAttribute(String name)**
returns true if an attribute with the given name is specified on this element; otherwise, it returns false
- **String getAttribute(String name)**
returns the name of the given attribute as a string, or the empty string if that attribute does not have a specified value
- **void setAttribute(String name, String value) throws DOMException**
adds a new attribute; if an attribute with the given name is already present in the element, its value is simply changed

Element Interface

- `void removeAttribute(String name)` throws `DOMException`
removes the attribute with the given name
- `Attr getAttributeNode(String name)`
returns an attribute node with the specified name, or null if such an attribute does not exist
- `Attr setAttributeNode(Attr newAttr)` throws `DOMException`
adds a new attribute node; if the specified attribute exists, then the replaced attribute node is returned
- `Attr removeAttributeNode(Attr oldAttr)` throws `DOMException`
removes the specified attribute node, and returns it

see <http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Element.html>

Attribute Interface

- `String getName()`
returns the name of the attribute
- `String getValue()`
returns the value of the attribute as a string
- `Element getOwnerElement()`
the element this attribute is attached to

see <http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/Attr.html>