3.2 Document Object Model (DOM)

- How to provide uniform access to structured documents in diverse applications (parsers, browsers, editors, databases)?
- Overview of W3C DOM Specification
  - second one in the "XML-family" of recommendations
    - Level 1, W3C Rec. Oct. 1998
    - Level 2, W3C Rec. Nov. 2000
    - Level 3, W3C Working Draft (January 2002)
- What does DOM specify, and how to use it?

DOM: What is it?

- An object-based, language-neutral API for XML and HTML documents
  - allows programs and scripts to build documents, navigate their structure, add, modify or delete elements and content
  - Provides a foundation for developing querying, filtering, transformation, rendering etc. applications on top of DOM implementations
- In contrast to "Serial Access XML," could think as "Directly Obtainable in Memory"

DOM structure model

- Based on O-O concepts:
  - methods (to access or change object's state)
  - interfaces (declaration of a set of methods)
  - objects (encapsulation of data and methods)
- Roughly similar to the XSLT/XPath data model (to be discussed later)
  - a parse tree
  - Tree-like structure implied by the abstract relationships defined by the programming interfaces;
    Does not necessarily reflect data structures used by an implementation (but probably does)

Structure of DOM Level 1

I: DOM Core Interfaces
- Fundamental interfaces
  - basic interfaces to structured documents
  - Extended interfaces
    - XML specific: CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction

II: DOM HTML Interfaces
- more convenient to access HTML documents
- (we ignore these)

DOM Level 2

- Level 1: basic representation and manipulation of document structure and content
  (No access to the contents of a DTD)
- DOM Level 2 adds
  - support for namespaces
  - accessing elements by ID attribute values
  - optional features
    - interfaces to document views and style sheets
    - an event model (for, say, user actions on elements)
    - methods for traversing the document tree and manipulating regions of document (e.g., selected by the user of an editor)
  - Loading and writing of docs not specified (\rightarrow Level 3)
**DOM Language Bindings**

- **Language-independence:**
  - DOM interfaces are defined using OMG Interface Definition Language (IDL; Defined in Corba Specification)
- **Language bindings** (implementations of DOM interfaces) defined in the Recommendation for
  - Java and
  - ECMAScript (standardised JavaScript)

**Core Interfaces: Node & its variants**

**Object Creation in DOM**

- Each DOM object X lives in the context of a Document: X.getOwnerDocument()
- Objects implementing interface X are created by factory methods
  
  \[ D . createX ( . . ) ; \]

  where D is a Document object. E.g:
  - `createElement("A")`, `createAttribute("href")`, `createTextNode("Hello!"")`
- Creation and persistent saving of Documents left to be specified by implementations

**DOM interfaces: Node**

**DOM interfaces: Document**

**Extended Interfaces**

**ProcessingInstruction**

**EntityReference**
Accessing properties of a Node

- `Node.getNodeName()`
  - for an `Element` = `getTagName()`
  - for an `Attr` = the name of the attribute
  - for `Text` = `#text` etc
- `Node.getNodeValue()`
  - content of a text node, value of attribute, ...
  - `null` for an `Element` (in XSLT/Xpath: the full textual content)
- `Node.getNodeType()`
  - numeric constants (1, 2, 3, ..., 12) for `ELEMENT_NODE`, `ATTRIBUTE_NODE`, `TEXT_NODE`, ...

Content and element manipulation

- **Manipulating CharacterData D:**
  - `D.substringData(offset, count)`
  - `D.appendData(string)`
  - `D.insertData(offset, string)`
  - `D.deleteData(offset, count)`
  - `D.replaceData(offset, count, string)` (= delete + insert)
- **Accessing attributes of an `Element` object E:**
  - `E.getAttribute(name)`
  - `E.setAttribute(name, value)`
  - `E.removeAttribute(name)`

Additional Core Interfaces (1)

- `NodeList` for ordered lists of nodes
  - e.g. from `Node.getChildNodes()` or `Element.getElementsByTagName("name")`
  - all descendant elements of type "name" in document order (wild-card "*" matches any element type)
- **Accessing a specific node, or iterating over all nodes of a `NodeList`:**
  - E.g. Java code to process all children:
    ```java
    for (i=0; i<node.getChildNodes().getLength(); i++)
      process(node.getChildNodes().item(i));
    ```

Additional Core Interfaces (2)

- `NamedNodeMap` for unordered sets of nodes accessed by their name:
  - e.g. from `Node.getAttributes()`
- `NodeLists` and `NamedNodeMaps` are "live":
  - changes to the document structure reflected to their contents

DOM: Implementations

- **Java-based parsers**
  - e.g. IBM XML4J, Apache Xerces, Apache Crimson
- **MS IE5 browser:** COM programming interfaces for C/C++ and MS Visual Basic, ActiveX object programming interfaces for script languages
- **XML::DOM** (Perl implementation of DOM Level 1)
- **Others? Non-parser-implementations?**
  (Participation of vendors of different kinds of systems in DOM WG has been active.)

A Java-DOM Example

- **A stand-alone toy application BuildXml**
  - either creates a new `db`-document with two `person` elements, or adds them to an existing `db` document
  - based on the example in Sect. 8.6 of Deitel et al: XML - How to program
- **Technical basis**
  - `DOM` support in Sun JAXP
  - native XML document initialisation and storage methods of the JAXP 1.1 default parser (Apache Crimson)
Class for modifying the document in file fileName:

```java
public class BuildXml {
    private Document document;

    public BuildXml(String fileName) {
        File docFile = new File(fileName);
        Element root = null; // doc root element

        // Obtain a SAX-based parser:
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();

        try {
            // to get a new DocumentBuilder:
            DocumentBuilder builder =
                factory.newInstance();

            if (!docFile.exists()) { //create new doc
                document = builder.newDocument();
                // add a comment:
                Comment comment =
                    document.createComment(
                        "A simple personnel list");
                document.appendChild(comment);
            } else { // access an existing doc
                document = builder.parse(docFile);
                root = document.getDocumentElement();
            }
        }

        root = document.createElement("db");
        document.appendChild(root);

        // Create and add two child elements to root:
        Node personNode =
            createPersonNode(document, "1234",
                             "Pekka", "Kilpeläinen");
        root.appendChild(personNode);
        personNode =
            createPersonNode(document, "5678",
                             "Irma", "Könönen");
        root.appendChild(personNode);

        try { // to write the
            ((XmlDocument) document).write( new FileOutputStream(fileName));
        } catch ( IOException ioe ) {
            ioe.printStackTrace();
        }
    }
}
```

Finally, store the result document:

```java
try {
    XML document to file fileName
    (XmlDocument) document).write( new FileOutputStream(fileName));
} catch ( IOException ioe ) {
    ioe.printStackTrace();
}
```
Subroutine to create `person` elements

```java
public Node createPersonNode(Document document, String idNum, String fName, String lName) {
    Element person = document.createElement("person");
    person.setAttribute("idnum", idNum);
    Element firstName = document.createElement("first");
    person.appendChild(firstName);
    firstName.appendChild(document.createTextNode(fName));
    /* ... similarly for a lastName */
    return person;
}
```

The main routine for `BuildXml`

```java
public static void main(String args[]) {
    if (args.length > 0) {
        String fileName = args[0];
        BuildXml buildXml = new BuildXml(fileName);
    } else {
        System.err.println("Give filename as argument");
    }
} // main
```

Summary of XML APIs

- **XML processors** make the structure and contents of XML documents available to applications through APIs
- **Event-based APIs**
  - notify application through parsing events
  - e.g., the SAX callback interfaces
- **Object-model (or tree) based APIs**
  - provide a full parse tree
  - e.g., DOM, W3C Recommendation
  - more convenient, but may require too much resources with the largest documents
- **Major parsers** support both SAX and DOM