3.2 Document Object Model (DOM)

- How access structured documents uniformly in parsers, browsers, editors, databases, etc.?
- Overview of the W3C DOM Spec
  - Level 1, W3C Rec, Oct. 1998
  - Level 2, W3C Rec, Nov. 2000
  - Level 3 Validation, Core, and Load and Save Recommendations (Spring 2004)
  - W3C DOM Activity has been closed

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) = syntax tree
  - Tree structure implied by abstract relationships defined by the API. Data structures of an implementation may differ

Structure of DOM Level 1

I: DOM Core Interfaces
- Fundamental interfaces
  - basic interfaces: Document, Element, Attr, Text, ...
  - "Extended" (XML specific) interfaces
    - CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction

II: DOM HTML Interfaces
- more convenient access to HTML documents
- (we'll 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 (we'll skip these)
    - 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)
  - Load/Save of documents not specified (until 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 interfaces) defined in the Recommendation for
  - Java (See the Java API doc) and
  - ECMAScript (standardised JavaScript)

Core Interfaces: Node & its variants
The Value of a Node

- `node.getNodeValue()`: content of a text node, value of an attribute, ...
- null for an `Element` (null)
- (in XSLT/XPath the value of a node is its full textual content)
- DOM 3 gives access to full textual content with the method `node.getTextContent()`

Object Creation in DOM

- Each DOM Node `n` belongs to a `Document`:
  - `n.getOwnerDocument()`
- Objects implementing interface `X` are created by factory methods:
  - `Document.createX(...)`
  - E.g: when doc is a `Document object`
    - `doc.createElement("A")`, `doc.createAttribute("href")`, `doc.createTextNode("Hello!")`
- Loading & saving specified in DOM3 (or via implementation-specific methods, or JAXP)

Text Content Manipulation in DOM

- For an object `c` that implements the `CharacterData` interface:
  - `(Text, Comments, CDATASections)`:
    - `c.substringData(offset, count)`
    - `c.appendData(string)`
    - `c.insertData(offset, string)`
    - `c.deleteData(offset, count)`
    - `c.replaceData(offset, count)`

Additional Core Interfaces (1)

- `NodeList` for ordered lists of nodes
  - from `Node.getChildNodes()` or `Element.getElementsByTagName("name")`
  - (a proper) descendant elements of type "name" in document order ("*" = any element type)
- Accessing a specific node, or iterating over all nodes of a `NodeList`:
  - E.g., to process all children of `node`:
    - `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. `Node.getAttribute()`
- **NodeLists** and **NamedNodeMaps** are "live":
  - updates of the document are reflected to their contents
  - e.g.: this would delete other child of node `n`:
    ```
    Node eList = n.getChildNodes();
    for (int i = 0; i < eList.getLength(); i++)
      n.removeChild(eList.item(i));
    ``
  - That's strange! (What happens?)

DOM: XML Implementations

- **Java-based parsers**
  - e.g.: Apache Xerces, Apache Crimson, ...
- In MS IE browser: COM programming interfaces for C/C++ and Visual Basic: ActiveX object programming interfaces for script languages
- **Perl**: `XML::DOM (Implements DOM Level 1)`
- **Others**: APIs for other applications than parsers?
  - Vendors of different kinds of systems have participated in the W3C DOM WG

A Java-DOM Example

- Command-line tool `RegListMgr` for maintaining a course registration list
  - with single-letter commands for listing, adding, updating and deleting student records
- **Example:**
  ```
  $ java RegListMgr reglist.xml
  Document loaded successfully
 .list the contents
  40: Tero Ulvinen, TKM1, tero@fake.addr.fi, 2
  41: Juho Vilinmaen, TTK, juho@fake.addr.fi, 1
  ``

Registration List: the XML file

```
<reglist lastID="41">
  <student id="RDK1">
    <name>juho@fake.addr.fi</name>
    <family>Abopalo</family></student>
</reglist>
```

Loading and Saving the RegList

- **Loading** of the registration list into DOM Document `doc` implemented with a JAXP `DocumentBuilder`
  - assume this has been done: `doc` is a handle to the Document
- **Saving** implemented with a JAXP `Transformer`
  - ... to be discussed later

Listing student records (1)

```java
NodeList students =
  doc.getElementsByTagName("student");
for (int i = 0; i < students.getLength(); i++)
  showStudent(students.item(i));

private void showStudent(Element student) {
  // Collect relevant sub-elements:
  Node given =
    student.getElementsByTagName("given").item(0);
  Node family =
    given getNextSibling();
  Node branchAndYear =
    student.getElementsByTagName("branchAndYear").item(0);
  Node email =
    bAndY.getNextSibling();
  Node group =
    email.getNextSibling();

  // value of family, branchAndYear, email, and group
}
```

Listing student records (2)

```java
// Method showStudent continues:
System.out.print(
  student.getAttribute("id").substring(3));
System.out.print("":
  given.getFirstChild().getNodeValue();
  // or given.getTextContent() with DOM3
  // ... similarly access and display the
  // value of family, branchAndYear, email, and group
  // ...
}
```

Registration list: the DTD

```xml
<!ELEMENT reglist (student*)>
<!ATTLIST reglist
  lastID CDATA #REQUIRED >
<!ELEMENT student (name, branchAndYear, email, group)>
<!ATTLIST student
  id ID #REQUIRED >
<!ELEMENT name (given, family)>
<!ELEMENT given [#PCDATA]>
<!ELEMENT branchAndYear [text]>
```

Saving implemented with a JAXP `Transformer`

- **Others?** APIs for other applications than parsers?
- Vendors of different kinds of systems have participated in the W3C DOM WG
Adding New Records

Example:

```java
Element rootElem = doc.getDocumentElement();
String lastID = rootElem.getAttribute("lastID");
int lastIDnum = java.lang.Integer.parseInt(lastID);
System.out.println("First name (or <return> to finish): ");
String firstName = terminalReader.readLine().trim();
System.out.print( "Last name: ");
String lastName = terminalReader.readLine().trim();
System.out.print( "Branch&year: ");
String bAndY = terminalReader.readLine().trim();
System.out.print( "Email: ");
String email = terminalReader.readLine().trim();
System.out.print( "Group: ");
String grp = terminalReader.readLine().trim();

while (firstName.length() > 0) {
    // Get the next unused ID:
    ID = "RDK" + new Integer(++lastIDnum).toString();
    // and group from the terminal, and then ... 
    // … Read values lastName, bAndY, email, 
    // and group from the terminal, and then ... 
}
```

Creating new student records (1)

```java
private Element
    newStudent(Document doc, String ID, String fName, String lName, String bAndY, String email, String grp) {
        Element stu = doc.createElement("student");
        stu.setAttribute("id", ID);
        Element newName = doc.createElement("given");
        newName.appendChild(doc.createTextNode(createTextNode(fName)));
        stu.appendChild(newName);
        Element newEmail = doc.createElement("email");
        newEmail.appendChild(doc.createTextNode(createTextNode(email)));
        stu.appendChild(newEmail);
        Element newGrp = doc.createElement("group");
        newGrp.appendChild(doc.createTextNode(createTextNode(bAndY)));
        stu.appendChild(newGrp);
        return stu;
    }
```

Creating new student records (2)

```java
// method newStudent( ) continues:
    newGrp.appendChild(doc.createTextNode(createTextNode(bAndY)));
    stu.appendChild(newGrp);
    Element newEmail = doc.createElement("email");
    newEmail.appendChild(doc.createTextNode(email));
    stu.appendChild(newEmail);
    Element newName = doc.createElement("given");
    newName.appendChild(doc.createTextNode(fName));
    stu.appendChild(newName);
    return stu;
}
```

Implementing addition of records (1)

```java
public void
    addStudents() {
        Element rootElem = doc.getDocumentElement();
        String lastID = rootElem.getAttribute("lastID");
        int lastIDnum = java.lang.Integer.parseInt(lastID);
        System.out.print("First name (or <return> to finish): ");
        String firstName = terminalReader.readLine().trim();
        // while firstName.length() > 0 {
        // Get the next unused ID:
        ID = "RDK" + new Integer(++lastIDnum).toString();
        // and group from the terminal, and then ... 
        // … Read values lastName, bAndY, email, 
        // and group from the terminal, and then ... 
    }
```

Updating and Deletions

- Updates and deletions implemented similarly, by manipulating the DOM structures
- To be treated in the exercises

Summary of XML APIs so far

- Give applications access to the structure and contents of XML documents
- Event-based APIs (e.g. SAX)
  - notify application through parsing events
  - efficient
- Object-model (or tree) based APIs (e.g. DOM)
  - provide a full parse tree
  - more convenient, but require much resources with large documents
- Major parsers support both SAX and DOM
  - used through proprietary methods
  - used through JAXP ( -> next)