3.2 Document Object Model (DOM)

- How access structured documents uniformly in parsers, browsers, editors, databases, ...?
- Overview of the W3C DOM Spec
  - Level 1: W3C Rec, Oct. 1998
  - Level 2, W3C Rec, Nov. 2000
  - Level 3 in progress (as 21 modules); Validation, Core, and Load and Save Recommendations (Spring 2004)

DOM: What is it?

- An object-based, language-neutral API for XML and HTML documents
  - Allows programs and scripts to build, access, and modify documents
  - Supports the development of querying, filtering, transformation, formatting 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) = syntax tree
  - Tree structure implied by abstract relationships defined by the API; Data structures of an implementation may differ (but hardly do(?)

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

Node
  - Document
  - DocumentFragment
  - Element
  - Attr
CharacterData
  - Text
  - CDATASection
Comment
pragma
  - NamedNodeMap
DocumentType
  - Notation
  - Entity
EntityReference
  - ProcessingInstruction
**Type and Name of a Node**

- `node.getNodeType()`: short int constants 1, 2, ..., 12 for `Node.ELEMENT_NODE`, `Node.ATTRIBUTE_NODE`, `Node.TEXT_NODE`, ...
- `node.getNodeName()`: – for an `Element` = `node.getElementsByTagName()` – for an `Attr`: the name of the attribute – for anonymous nodes: "#text", "#document", "#comment" etc.

**Object Creation in DOM**

- Each DOM Node \( n \) belongs to a `Document`: \( n.\text{getOwnerDocument}() \)
- Objects implementing interface \( X \) are created by factory methods `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):
  - \( c\text{-substringData}(offset, count) \)
  - \( c\text{-appendData}(string) \)
  - \( c\text{-insertData}(offset, string) \)
  - \( c\text{-deleteData}(offset, count) \)
  - \( c\text{-replaceData}(offset, count, string) \)  

**Additional Core Interfaces (1)**

- `NodeList` for ordered lists of nodes – e.g. from `Node.getChildNodes()` or `Element.getElementsByTagName("name")` of all descendant elements of type "name" in document order (*"* matches any element type)
- Accessing a specific node, or iterating over all nodes of a `NodeList`: – E.g., `for (i=0; i<node.getChildNodes().getLength(); i++)`
Additional Core Interfaces (2)

- NamedNodeMap for unordered sets of nodes accessed by their name:
  - e.g., Node.getAttribute()
- NodeList and NamedNodeMap are "live":
  - updates of the document structure are reflected to their contents
  - e.g., this would delete every other child of node n:
    NodeList cList = n.getChildNodes();
    for (i=0; i<cList.getLength(); i++)
      n.removeChild(cList.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

Listing student records (1)

NodeList students =
  doc.getElementsByTagName("student");
for (int i=0; i<students.getLength(); i++)
  showStudent((Element) students.item(i));
private void showStudent(Element student) {
  // Collect relevant sub-elements:
  Node given =
    student.getElementsByTagName("given").item(0);
  Node family = given.getNextSibling();
  Node bAndY = student.getTagName()="branchAndYear").item(0);
  Node email = bAndY.getNextSibling();
  Node group = email.getNextSibling();

Listing student records (2)

// 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, bAndY, email, and group
}  
// showStudent

Registration List: the DTD

<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)>
</reglist>

Listing student records (2)

// Method showStudent continues:
System.out.print(" + " +  
  given.getFirstChild().getNodeValue();
}  
// or given.getTextContent() with DOM3
// similarly access and display the  
// value of family, bAndY, email, and group
}  
// showStudent
Adding New Records

- Example:

  ```java
da.add(students); // add students
```

First name (or <return> to finish): Antti
Last name: Ahkera
Branch&Year: tkt3
Email: antti@fake.addr.fi
Group: 2

First name (or <return> to finish): Finished adding records

```
41: helvi viinikainen, tkt5, helvi@fake.addr.fi, 1
42: Antti Ahkera, tkt3, antti@fake.addr.fi, 2
```

Implementing addition of records (1)

```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();
while (firstName.length() > 0) {
    // Get the next unused ID:
    ID = "RDK" + new Integer(++lastIDnum).toString();
    // … 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("name");
        Element newGiven = doc.createElement("given");
        newGiven.appendChild(doc.createTextNode(fName));
        Element newFamily = doc.createElement("family");
        newFamily.appendChild(doc.createTextNode(lName));
        newName.appendChild(newGiven);
        newName.appendChild(newFamily);
        stu.appendChild(newName);
        Element newBr = doc.createElement("branchAndYear");
        newBr.appendChild(doc.createTextNode(bAndY));
        stu.appendChild(newBr);
        Element newEmail = doc.createElement("email");
        newEmail.appendChild(doc.createTextNode(email));
        stu.appendChild(newEmail);
        Element newGrp = doc.createElement("group");
        newGrp.appendChild(doc.createTextNode(group));
        stu.appendChild(newGrp);
        return stu;
    }
```

Creating new student records (2)

```java
// method newStudent(.) continues:
Element newBr = doc.createElement("branchAndYear");
newBr.appendChild(doc.createTextNode(bAndY));
stu.appendChild(newBr);
Element newEmail = doc.createElement("email");
newEmail.appendChild(doc.createTextNode(email));
stu.appendChild(newEmail);
Element newGrp = doc.createElement("group");
newGrp.appendChild(doc.createTextNode(group));
stu.appendChild(newGrp);
return stu;
```

Updates 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)