XML PARSERS – DOM

UNIT-II

Document Object Model (DOM)

- DOM provides way of representing an XML document in memory - so manipulated by the software
- DOM provides API to access elements

- **DOM is not the following**
  - Not a mechanism for persisting, or *storing, objects as XML documents*
  - DOM is not a set of data structures
  - DOM does not specify what information in a document is relevant or how information should be structured
  - DOM is not a COM, CORBA, or other technologies that include the words *object model.*

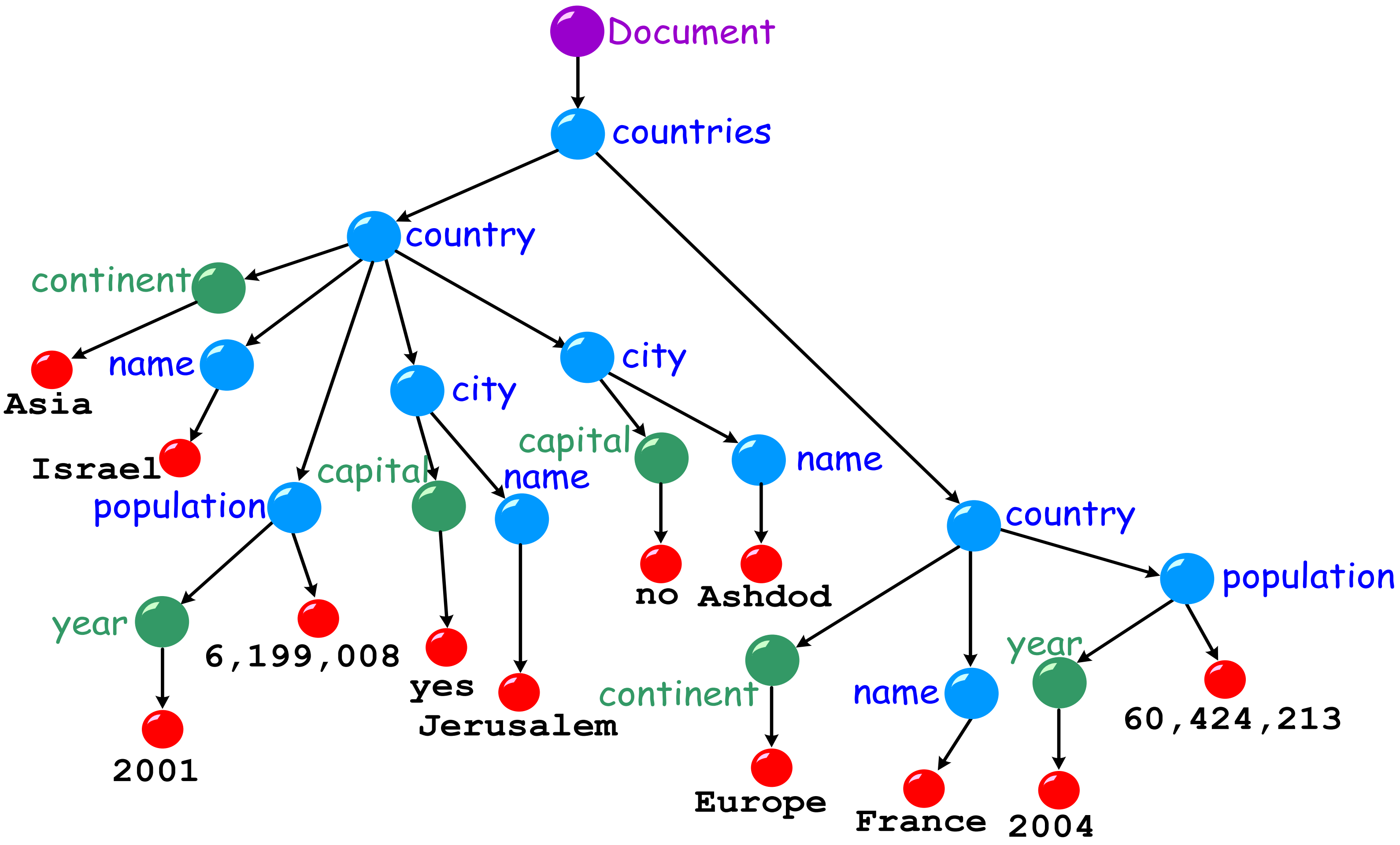
DOM Levels

- Till now three levels are in the works:
- Level 1 allows traversal of an XML document, manipulation of the content in that document
- Level 2 extends Level 1 with additional features such as namespace support, events, ranges, and so on
- Level 3 is currently a working draft

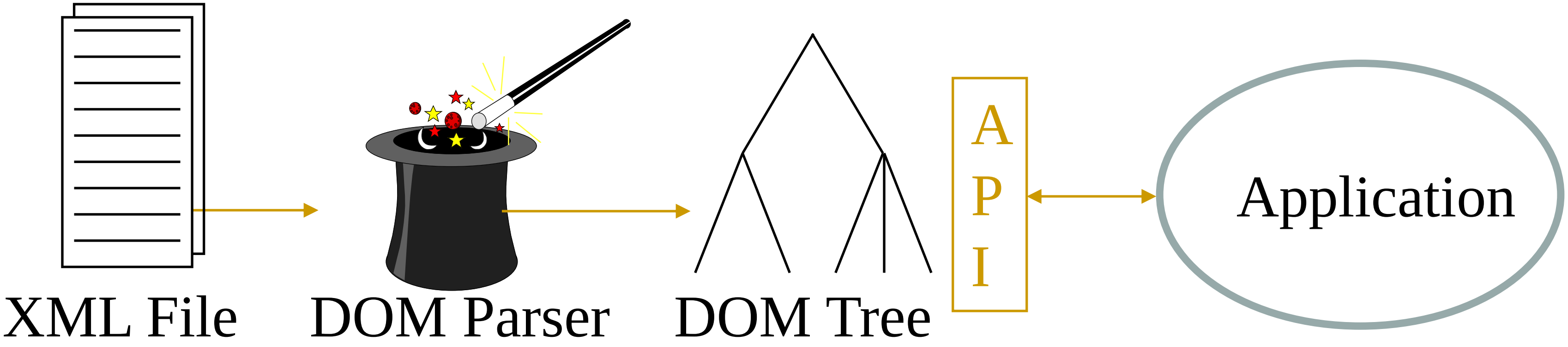
XML Document

```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6,199,008</population>
    <city capital="yes"><name>Jerusalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60,424,213</population>
  </country>
</countries>
```

DOM Tree


Document
countries
country
continent
Asia
name
Israel
population
year
2001
6,199,008
city
capital
yes
name
Jerusalem
city
capital
no
name
Ashdod
country
continent
Europe
name
France
population
year
2004
60,424,213

Using a DOM Tree



DOM Parser

- A DOM parser creates an internal tree structure in memory which is a ***DOM document*** object

- Client applications get the information of the original XML document by invoking methods on this ***Document*** object or on other objects it contains

JAVA Bindings

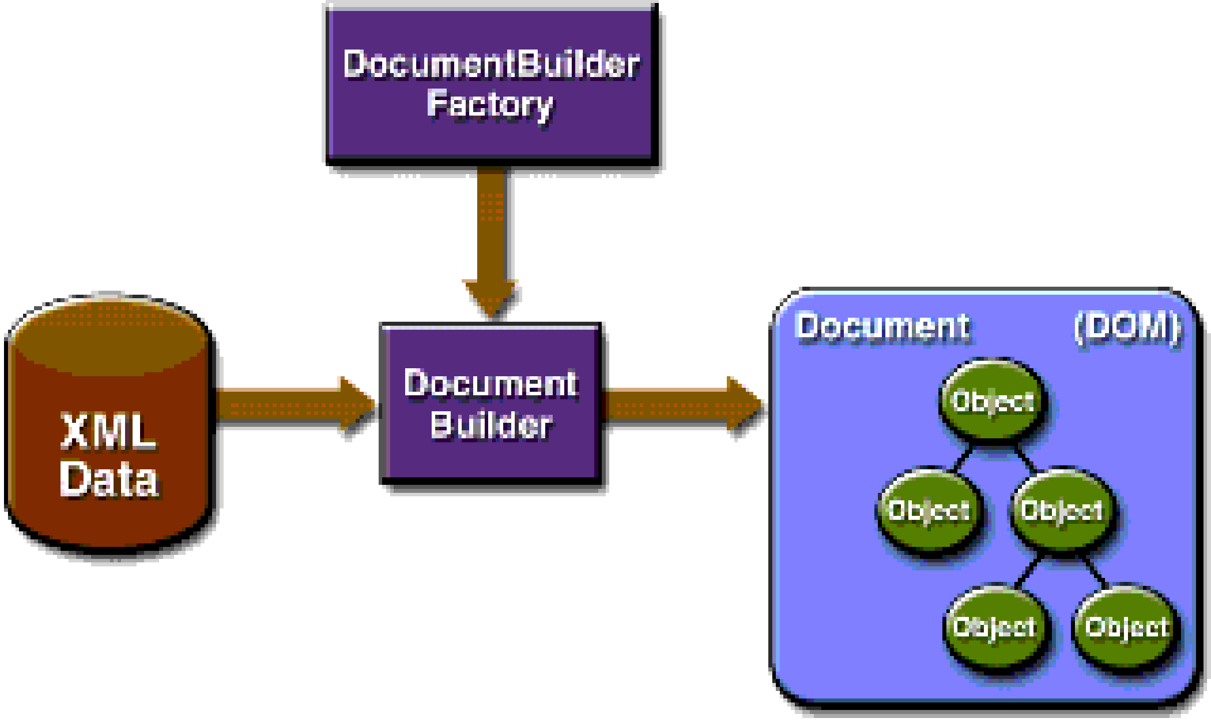
- DOM working group supplies Java language bindings as part of the DOM specification

- Two of the most popular:
  - Java APIs for XML Processing (JAXP), developed by Sun Microsystems
  - Xerces developed as part of the Apache XML project

- JAXP and Xerces are freely available is open source

DOM abstraction layer in Java -- architecture

- Allows vendors to supply their own DOM
- Implementation without requiring change to source code



Node Interface

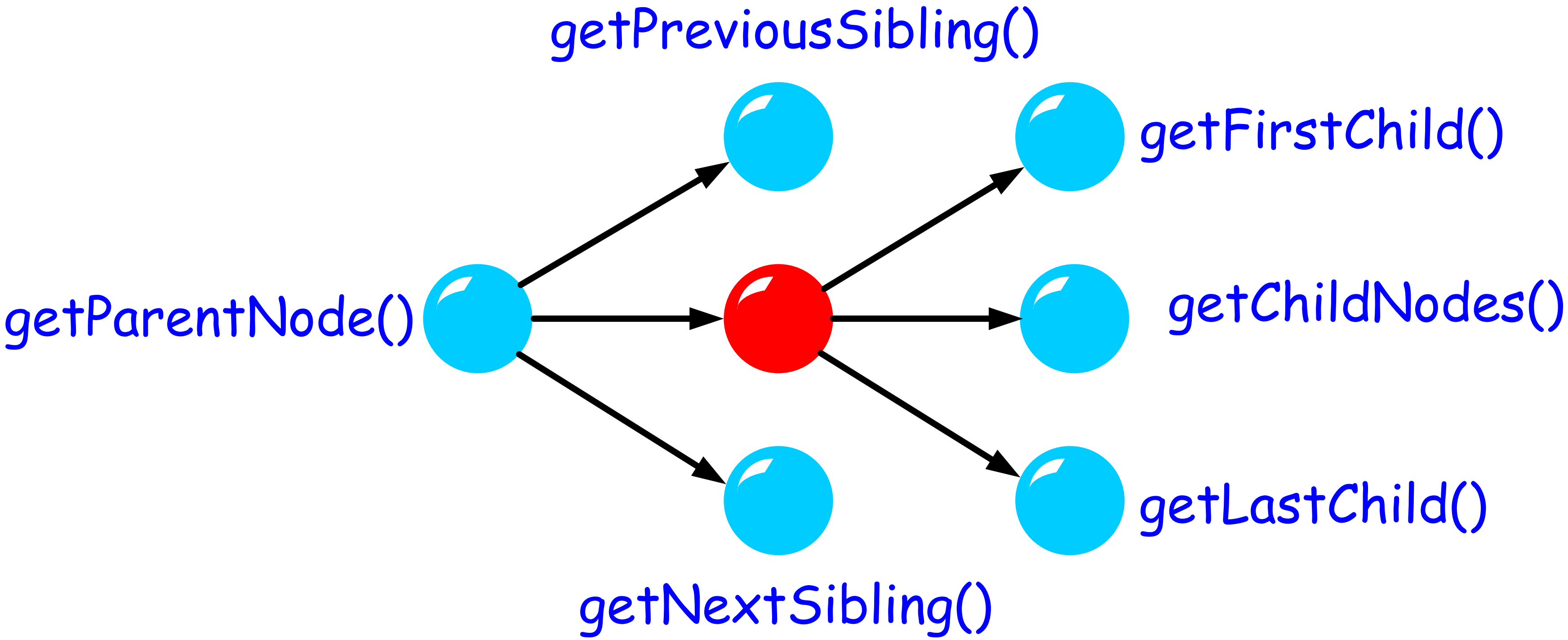
- The nodes of the DOM tree include
  - a special root (denoted *document*)
  - element nodes
  - text nodes and CDATA sections
  - attributes
  - Comments
  - DocumentType
  - Entity
  - EntityReference
  - Notation
  - ProcessingInstruction

- Every node in the DOM tree implements the **Node** interface

Node Navigation

- Every node has a specific location in tree
- `Node` interface specifies methods for tree navigation
  - `Node getFirstChild();`
  - `Node getLastChild();`
  - `Node getNextSibling();`
  - `Node getPreviousSibling();`
  - `Node getParentNode();`
  - `NodeList getChildNodes();`
  - `NamedNodeMap getAttributes()`

Node Navigation Contd…



An Example

- DocumentBuilderFactor factory = DocumentBuilderFactory.newInstance();

A factory instance
is the parser implementation.
Can be changed with runtime
System property

- /* set some factory options here */

From the factory one obtains
an instance of the parser

- DocumentBuilder builder = factory.newDocumentBuilder();

xmlFile can be an java.io.File,
an inputstream, etc.

- Document doc = builder.parse(xmlFile);

Example Contd…

- // Make a copy of the element subtree suitable for inserting into doc
- Node node = doc.importNode(element, true);
- // Get the parent Node
- parent = node.getParentNode();
- // Get children NodeList
- children = node.getChildNodes();
- // Get first child; null if no children
- Node child = node.getFirstChild();
- // Get last child; null if no children
- child = node.getLastChild();

Example Contd…

- // Get next sibling; null if node is last child
- Node sibling = node.getNextSibling();
- // Get previous sibling; null if node is first child
- sibling = node.getPreviousSibling();
- // Get first sibling
- sibling = node.getParentNode().getFirstChild();
- // Get last sibling sibling = node.getParentNode().getLastChild(); }
- }

Contd…

- System.out.println("Root element :" + doc.getDocumentElement().getNodeName());
- if (doc.hasChildNodes()) { printNote(doc.getChildNodes()); }
- } catch (Exception e) { System.out.println(e.getMessage()); } }

- private static void printNote(NodeList nodeList) {
- for (int count = 0; count < nodeList.getLength(); count++) {
- Node tempNode = nodeList.item(count); // make sure it's element node
- if (tempNode.getNodeType() == Node.ELEMENT_NODE) { // get node name and value
- System.out.println("\nNode Name =" + tempNode.getNodeName() + " [OPEN]");

Contd...

- System.out.println("Node Value =" +tempNode.getTextContent());
- if (tempNode.hasAttributes()) { // get attributes names and values
- NamedNodeMap nodeMap = tempNode.getAttributes();
- for (int i = 0; i < nodeMap.getLength(); i++) {
- Node node = nodeMap.item(i);
- System.out.println("attr name : " + node.getNodeName()); System.out.println("attr value : " + node.getNodeValue()); } }
- if (tempNode.hasChildNodes()) { // loop again if has child nodes printNote(tempNode.getChildNodes()); }
- System.out.println("Node Name =" + tempNode.getNodeName() + " [CLOSE]"); }

}

DOM Traversal and Range

- Supported in DOM Level2
- Can check the support using hasFeature( )
- Traversal - is a convenient way to walk through a DOM tree and select specific nodes
- Allows to find certain elements and perform operations on them
- **Traversal Interfaces**
- NodeIterator - Represents a subtree as a linear list and walk through nodes linearly

Contd…

- TreeWalker - Represents a subtree as a tree view
- NodeFilter - used with NodeIterator and TreeWalker to select specific nodes
- DocumentTraversal - Contains methods to create NodeIterator and TreeWalker instances

Example of NodeIterator

- NodeIterator iter =
- ((DocumentTraversal)document).createNodeIterator(
- root, NodeFilter.SHOW_ELEMENT,
- new NameNodeFilter(“book”), true);

- Node n = iter.nextNode();
- while (n != null) {
- System.out.println(n.getFirstChild().getNodeValue());
- n = iter.nextNode();
- }

Range Interface

- Range - This interface describes a range and contains methods to define, delete, insert content
- DocumentRange - This interface creates a range
- Support to range interface is tested by calling the hasFeature(…) method of the DOMImplementation interface

Example:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
docBuilder = dbf.newDocumentBuilder();
DOMImplementation domImp = docBuilder.getDOMImplementation();
if (domImp.hasFeature(“Range”, “2.0”))
```

Sample Code

- Code for delete a range of content

- Range r = ((DocumentRange)document).createRange();
- r.selectNodeContents(root.getFirstChild());
- r.deleteContents();

Other Alternatives

- **JDOM**
- DOM is purely language independent, but JDOM is meant for only JAVA
- DOM as language independent more complex

- **Aim of JDOM**
  - should be straightforward for Java programmers.
  - should support easy and efficient document modification.
  - should hide the complexities of XML
  - should integrate with DOM and SAX.
  - should be lightweight and fast.
  - should solve most of the Java/XML problems with little effort when compare with DOM

JDOM vs DOM

- **JDOM**
  - class-based API
  - Classes encapsulates documents, elements, attributes, text etc. minimize downcast
  - does not parse XML but it build JDOM objects from a DOM tree
- **DOM**
  - interface-based API
  - DOM is a strict hierarchy based on a node, leads to lots of downcasts and so reduced performance

Small Implementation of DOM

- **To work with DOM on a PDA, a full-blown DOM implementation is heavyweight**
- Smaller, simpler alternatives are needed
- Example: NanoXML, TinyXML, kXML

- NanoXML - nonvalidating parser, looks a lot like DOM, but it's much smaller, a light version is less than 6KB

- TinyXML - It's only for reading an XML document, cannot create a document
  - Has one class, TinyParser, and one interface, ParsedXML.
  - Just call static method TinyParser to parse a stream, file, or URL
  - The uncompressed class files are about 16KB

- NanoXML - designed specifically for J2ME resource-constrained devices
  - The most sophisticated of the three small parsers