# Processing XML with Java

Representation and Management of Data on the Internet

## Parsers

- What is a parser?
  - **A program that analyses the grammatical structure of an input, with respect to a given formal grammar**
  - The parser determines how a sentence can be constructed from the grammar of the language by describing the atomic elements of the input and the relationship among them
- How should an XML parser work?

## XML-Parsing Standards

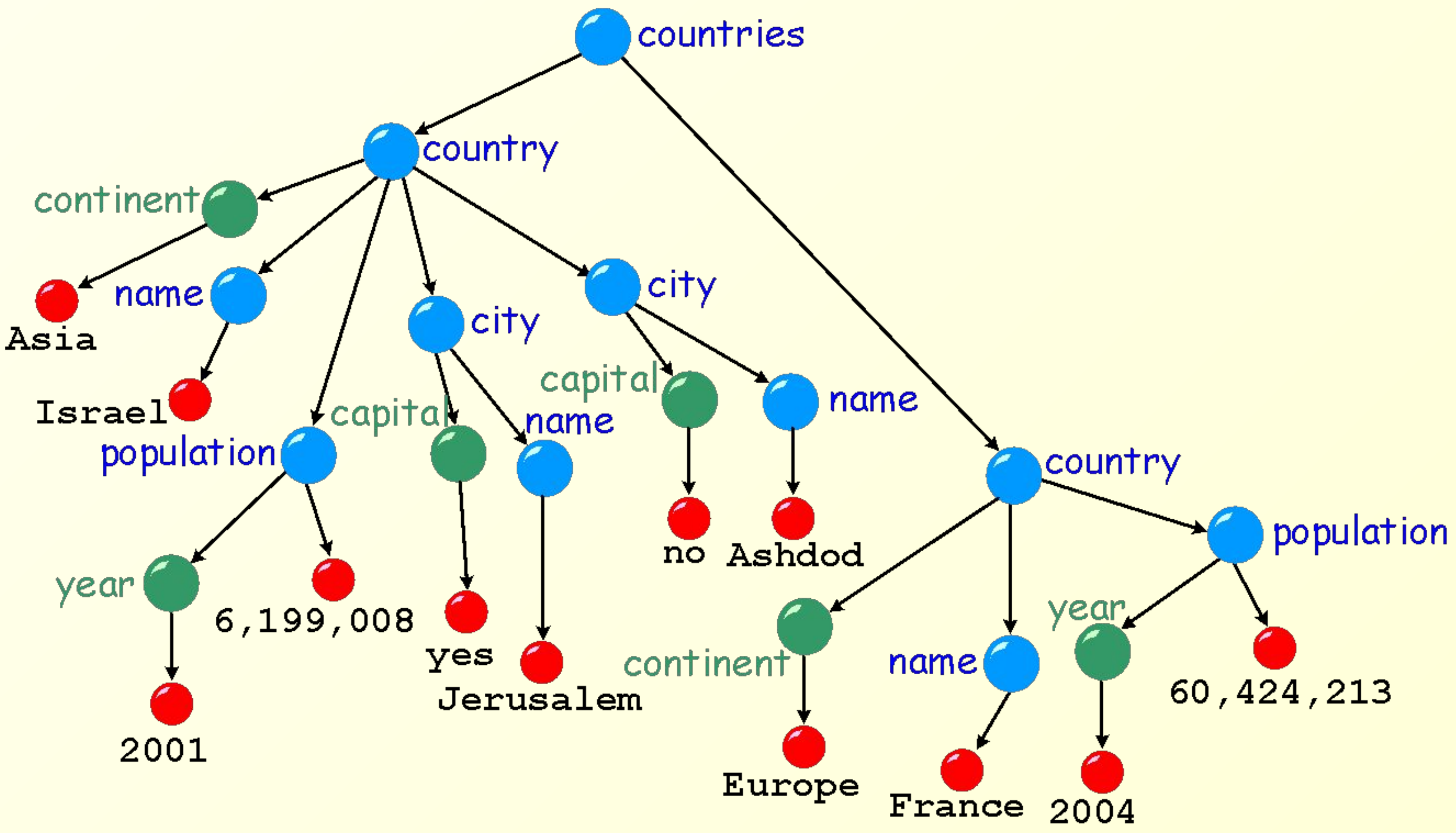
- We will consider two parsing methods that implement W3C standards for accessing XML
- **SAX**
  - event-driven parsing
  - "serial access" protocol
- **DOM**
  - convert XML into a tree of objects
  - "random access" protocol

## XML Examples

world.xml

```
<?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>Jerulsalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60,424,213</population>
  </country>
</countries>
```

## XML Tree Model



**world.dtd**

```
<!ELEMENT countries (country*)>
<!ELEMENT country (name,population?,city*)>
<!ATTLIST country continent CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (name)>
<!ATTLIST city capital (yes|no) "no">
<!ELEMENT population (#PCDATA)>
<!ATTLIST population year CDATA #IMPLIED>
<!ENTITY eu "Europe">
<!ENTITY as "Asia">
<!ENTITY af "Africa">
<!ENTITY am "America">
<!ENTITY au "Australia">
```

## *sales.xml*

```
<?xml version="1.0"?>
<forsale date="12/2/03"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <book>
    <title> <xhtml:em>DBI:</xhtml:em>
      <![CDATA[Where I Learned <xhtml>.]]>
    </title>
    <comment
      xmlns="http://www.cs.huji.ac.il/~dbi/comments">
      <par>My  <xhtml:b> favorite </xhtml:b> book!</par>
    </comment>
  </book>
</forsale>
```

**sales.xml**

```
<?xml version="1.0"?>
<forsale date="12/2/03"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <book>
    <title> <xhtml:h1> DBI </xhtml:h1>
      <![CDATA[Where I Learned <xhtml>.]]>
    </title>
    <comment
      xmlns="http://www.cs.huji.ac.il/~dbi/comments">
                                                    /par>

</b
</forsale>
```

Namespace: “http://www.w3.org/1999/xhtml”

Local name: “h1”

Qualified name: “xhtml:h1”

**_sales.xml_**

```
<?xml version="1.0"?>
<forsale date="12/2/03"
   xmlns:xhtml="http://www.w3.org/1999/xhtml">
   <book>
      <comment
         xmlns="http://www.cs.huji.ac.il/~dbi/comments">
        <par>My  <xhtml:b> favorite </xhtml:b> book!</par>
      </comment>
</book>
</forsale>
```

Namespace: “http://www.cs.huji.ac.il/~dbi/comments”

Local name: “par”

Qualified name: “par”

***sales.xml***

```
<?xml version="1.0"?>
<forsale date="12/2/03"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <book>
    <title>  <xhtml:h1>DBI</xhtml:h1>
      <![CDATA[Where I Learned <xhtml>.]]>
    </title>
    <comment
      xmlns="http://www.cs.huji.ac.il/~dbi/comments">
      <par>My  <xhtml:b> favorite </xhtml:b> book!</par>
    </comment>
</book>
</forsale>
```

Namespace: “”

Local name: “title”

Qualified name: “title”

## SAX – Simple API for XML

## SAX Parser

- SAX = Simple API for XML
- XML is read sequentially
- When a *parsing event* happens, the parser invokes the corresponding method of the corresponding handler
- The handlers are programmer's implementation of standard Java API (i.e., interfaces and classes)
- Similar to an I/O-Stream, goes in one direction

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

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

***Start Document***

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

**Start Element**

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

**Start Element**

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

**Comment**

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

***Start Element***

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

***Characters***

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

**End Element**

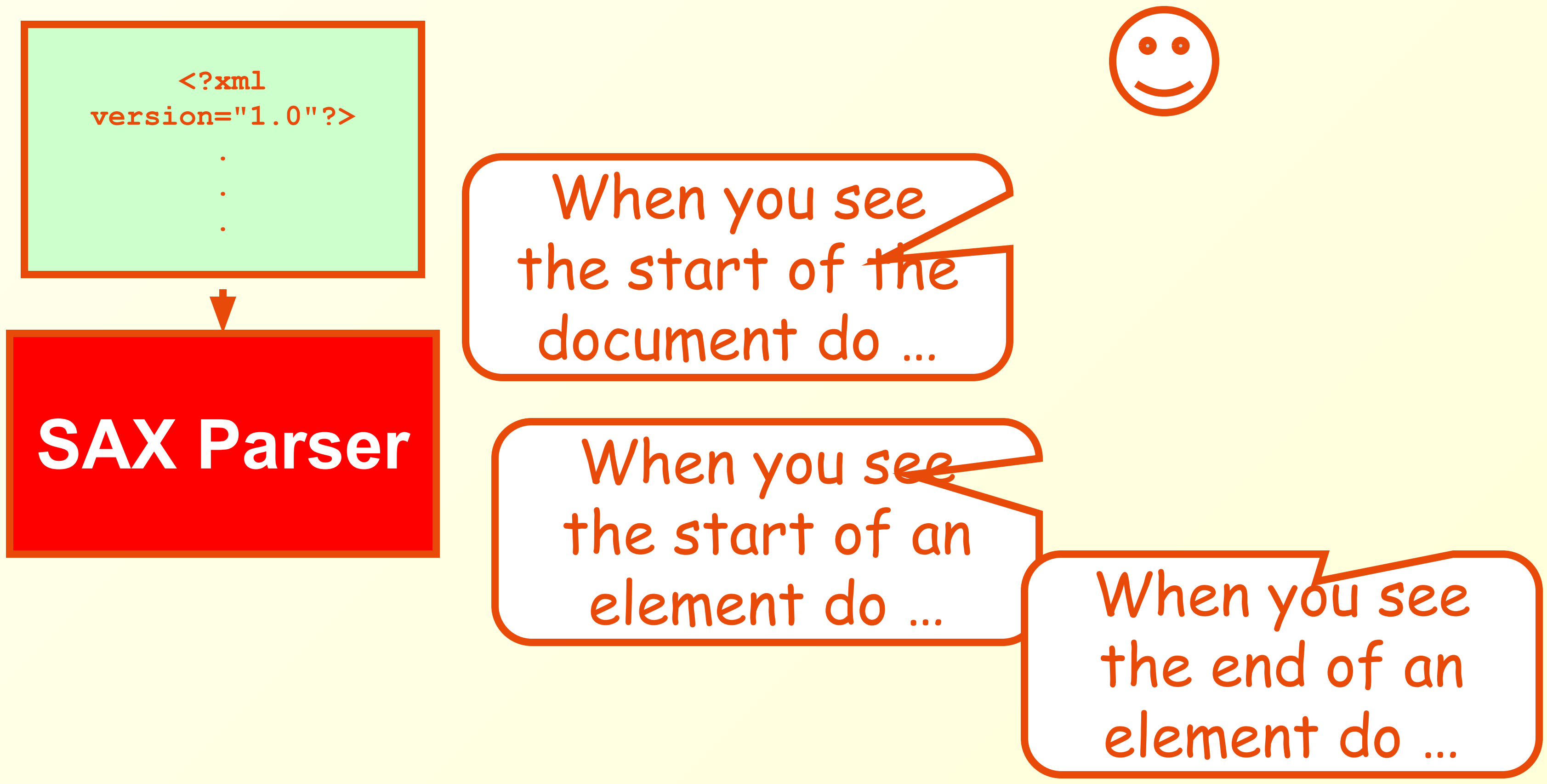
```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">  <!--israel-->
    <name>Israel</name>
    <population year="2001">6,199,008</population>
    <city capital="ye                alem</name></city>
    <city><name>Ashdod
  </country>
  <country continent="&eu;">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

**End Element**

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

**End Document**

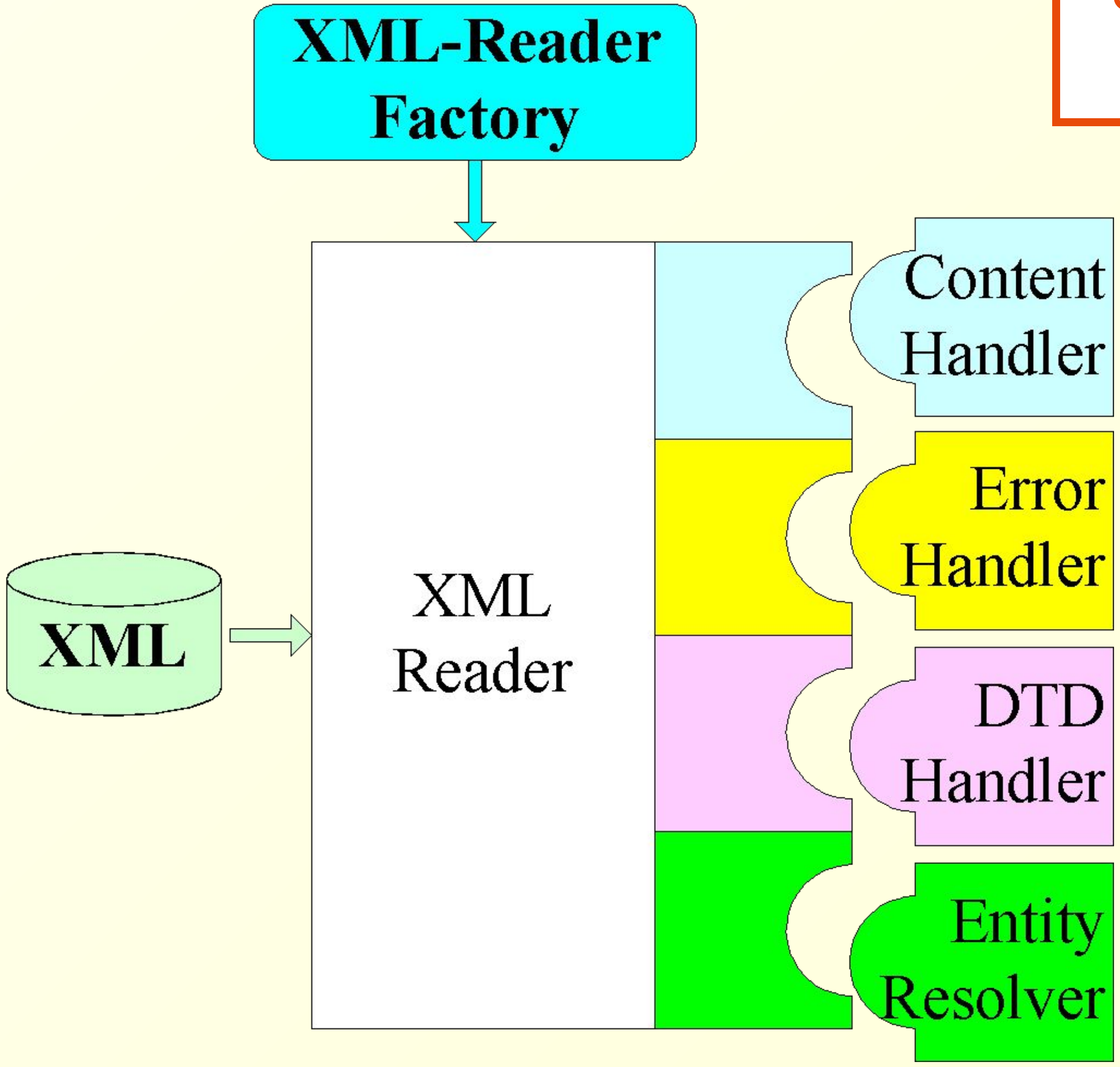
# SAX Parsers



<?xml version="1.0"?>
.
.
.
SAX Parser
When you see the start of the document do …
When you see the start of an element do …
When you see the end of an element do …

~~Used~~ to create a SAX Parser

Handles document events: start tag, end tag, etc.



Handles Parser Errors

Handles DTD

Handles Entities

## Creating a Parser

- The SAX interface is an accepted standard
- There are many implementations of many vendors
  - Standard API does not include an actual implementation, but Sun provides one with JDK
- Like to be able to change the implementation used without changing any code in the program
  - How is this done?

## Factory Design Pattern

- Have a “factory” class that creates the actual parsers
  - `org.xml.sax.helpers.XMLReaderFactory`
- The factory checks configurations, such as the of a system property, that specify the implementation
  - Can be set outside the Java code: a configuration file, a command-line argument, etc.
- In order to change the implementation, simply change the system property

## Creating a SAX Parser

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class EchoWithSax {
   public static void main(String[] args) throws Exception {
      System.setProperty("org.xml.sax.driver",
         "org.apache.xerces.parsers.SAXParser");
      XMLReader reader =
         XMLReaderFactory.createXMLReader();
      reader.parse("world.xml");
   }
}
```

## Implementing the Content Handler

- A SAX parser invokes methods such as `startDocument`, `startElement` and `endElement` of its *content handler* as it runs
- In order to react to parsing events we must:
  - implement the `ContentHandler` interface
  - set the parser's content handler with an instance of our `ContentHandler` implementation

## `ContentHandler` Methods

- `startDocument` - parsing begins
- `endDocument` - parsing ends
- `startElement` - an opening tag is encountered
- `endElement` - a closing tag is encountered
- `characters` - text (CDATA) is encountered
- `ignorableWhitespace` - white spaces that should be ignored (according to the DTD)
- and more ...

## The Default Handler

- The class `DefaultHandler` implements all handler interfaces (usually, in an empty manner)
  - i.e., `ContentHandler`, `EntityResolver`, `DTDHandler`, `ErrorHandler`
- An easy way to implement the `ContentHandler` interface is to extend `DefaultHandler`

## A Content Handler Example

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

public class EchoHandler extends DefaultHandler {

  int depth = 0;

  public void print(String line) {
    for(int i=0; i<depth; ++i) System.out.print("    ");
    System.out.println(line);
  }
```

# A Content Handler Example

```
public void startDocument() throws SAXException {
    print("BEGIN"); }
public void endDocument() throws SAXException {
    print("END"); }

public void startElement(String ns, String lName,
    String qName, Attributes attrs) throws SAXException {
    print("Element " + qName + "{");
    ++depth;
    for (int i = 0; i < attrs.getLength(); ++i)
      print(attrs.getLocalName(i) + "=" + attrs.getValue(i)); }
```

## A Content Handler Example

```
public void endElement(String ns, String lName,
   String qName) throws SAXException {
    --depth;
    print("}");  }

public void characters(char buf[], int offset, int len)
    throws SAXException {
    String s = new String(buf, offset, len).trim();
    ++depth; print(s); --depth; } }
```

## Fixing The Parser

```
public class EchoWithSax {
  public static void main(String[] args) throws Exception {
    XMLReader reader =
      XMLReaderFactory.createXMLReader();
    reader.setContentHandler(new EchoHandler());
    reader.parse("world.xml");
  }
}
```

## Empty Elements

- What do you think happens when the parser parses an empty element?

```
<rating stars="five" />
```

## `Attributes` Interface

- The **`Attributes`** interface provides an access to all attributes of an element
  - **`getLength()`**, **`getQName(i)`**, **`getValue(i)`**, **`getType(i)`**, **`getValue(qname)`**, etc.
- The following are possible types for attributes:
  CDATA, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES, NOTATION
- There is no distinction between attributes that are defined explicitly from those that are specified in the DTD (with a default value)

## `ErrorHandler` Interface

- We implement **`ErrorHandler`** to receive error events (similar to implementing **`ContentHandler`**)
- **`DefaultHandler`** implements **`ErrorHandler`** in an empty fashion, so we can extend it (as before)
- An **`ErrorHandler`** is registered with
  - **`reader.setErrorHandler(handler);`**
- Three methods:
  - **`void error(SAXParseException ex);`**
  - **`void fatalError(SAXParserExcpetion ex);`**
  - **`void warning(SAXParserException ex);`**

## Parsing Errors

- Fatal errors disable the parser from continuing parsing
  - For example, the document is not well formed, an unknown XML version is declared, etc.
- Errors occur the parser is validating and validity constrains are violated
- Warnings occur when abnormal (yet legal) conditions are encountered
  - For example, an entity is declared twice in the DTD

## EntityResolver and DTDHandler

- The class `EntityResolver` enables the programmer to specify a new source for translation of external entities
- The class `DTDHandler` enables the programmer to react to *notations* and *unparsed entities* declarations inside the DTD

# Features and Properties

- SAX parsers can be configured by setting their features and properties
- Syntax:
  - `reader.setFeature("`*`feature-url`*`", boolean)`
  - `reader.setProperty("`*`property-url`*`", Object)`
- Standard feature URLs have the form:

  `http://xml.org/sax/features/`*`feature-name`*
- Standard property URLs have the form

  `http://xml.org/sax/properties/`*`prop-name`*

## Feature/Property Examples

- Features:
  - **`namespaces`** - are namespaces supported?
  - **`validation`** - does the parser validate (against the declared DTD) ?
  - **http://apache.org/xml/features/nonvalidating/load-external-dtd**
    - Ignore the DTD? (spec. to Xerces implementation)
- Properties:
  - **`xml-string`** - the actual text that cased the current event (read-only)
  - **`lexical-handler`** - see the next slide...

## Lexical Events

- Lexical events have to do with the way that a document was written and not with its content
- Examples:
  - A comment is a lexical event (<!-- comment -->)
  - The use of an entity is a lexical event (&gt;)
- These can be dealt with by implementing the **`LexicalHandler`** interface, and setting the **`lexical-handler`** property to an instance of the handler

## `LexicalHandler` Methods

- comment(char[] ch, int start, int length)
- startCDATA()
- endCDATA()
- startEntity(java.lang.String name)
- endEntity(java.lang.String name)
- and more...

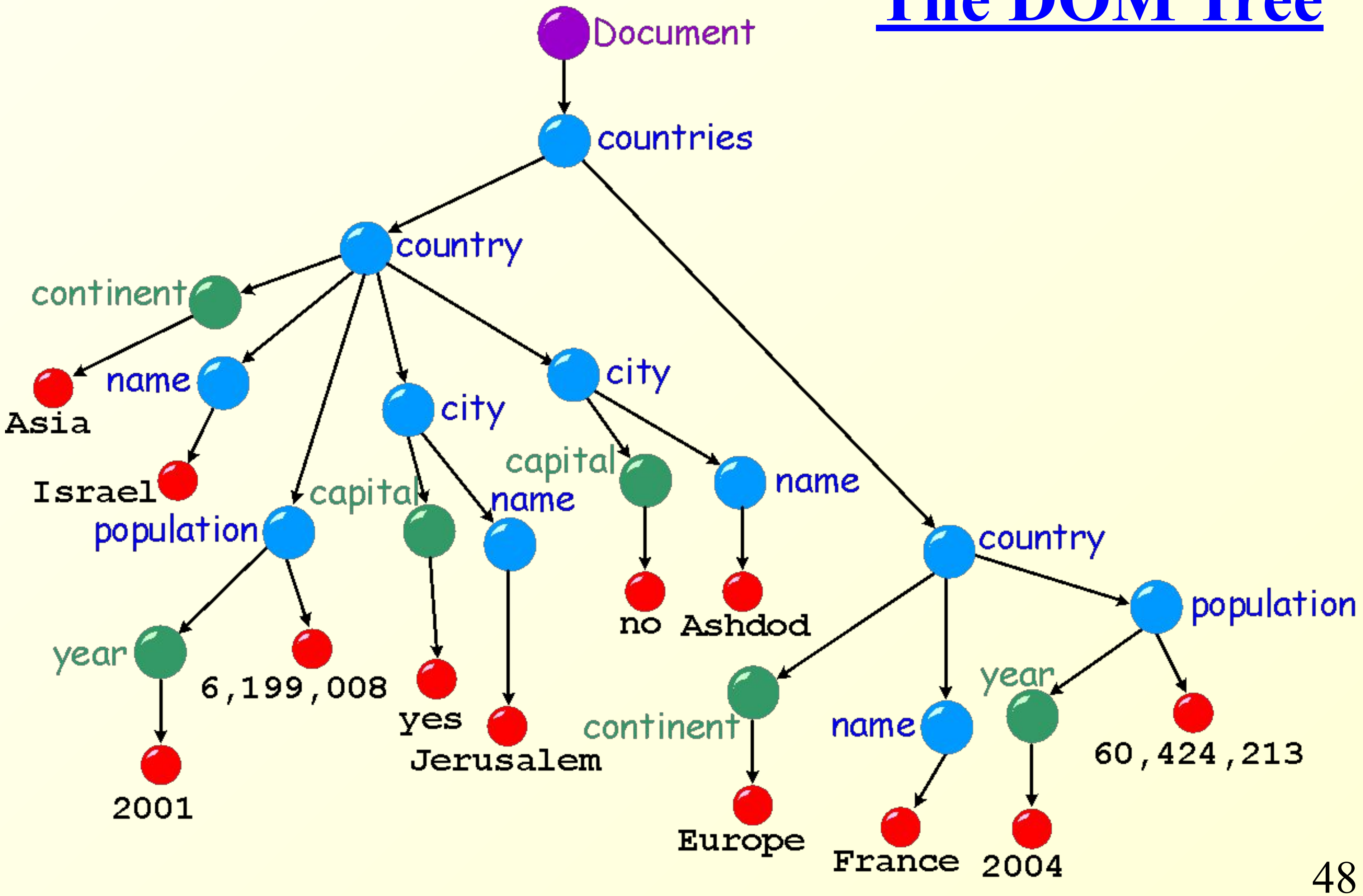
## DOM – Document Object Model

## DOM Parser

- DOM = Document Object Model
- Parser creates a tree object out of the document
- User accesses data by traversing the tree
  - The tree and its traversal conform to a W3C standard
- The API allows for constructing, accessing and manipulating the structure and content of XML documents

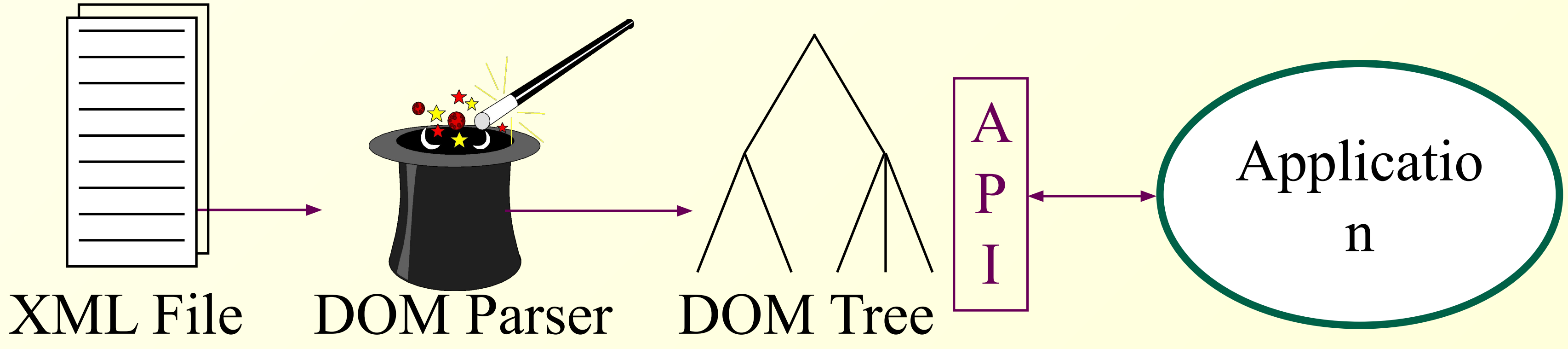
```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6,199,008</population>
    <city capital="yes"><name>Jerulsalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">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

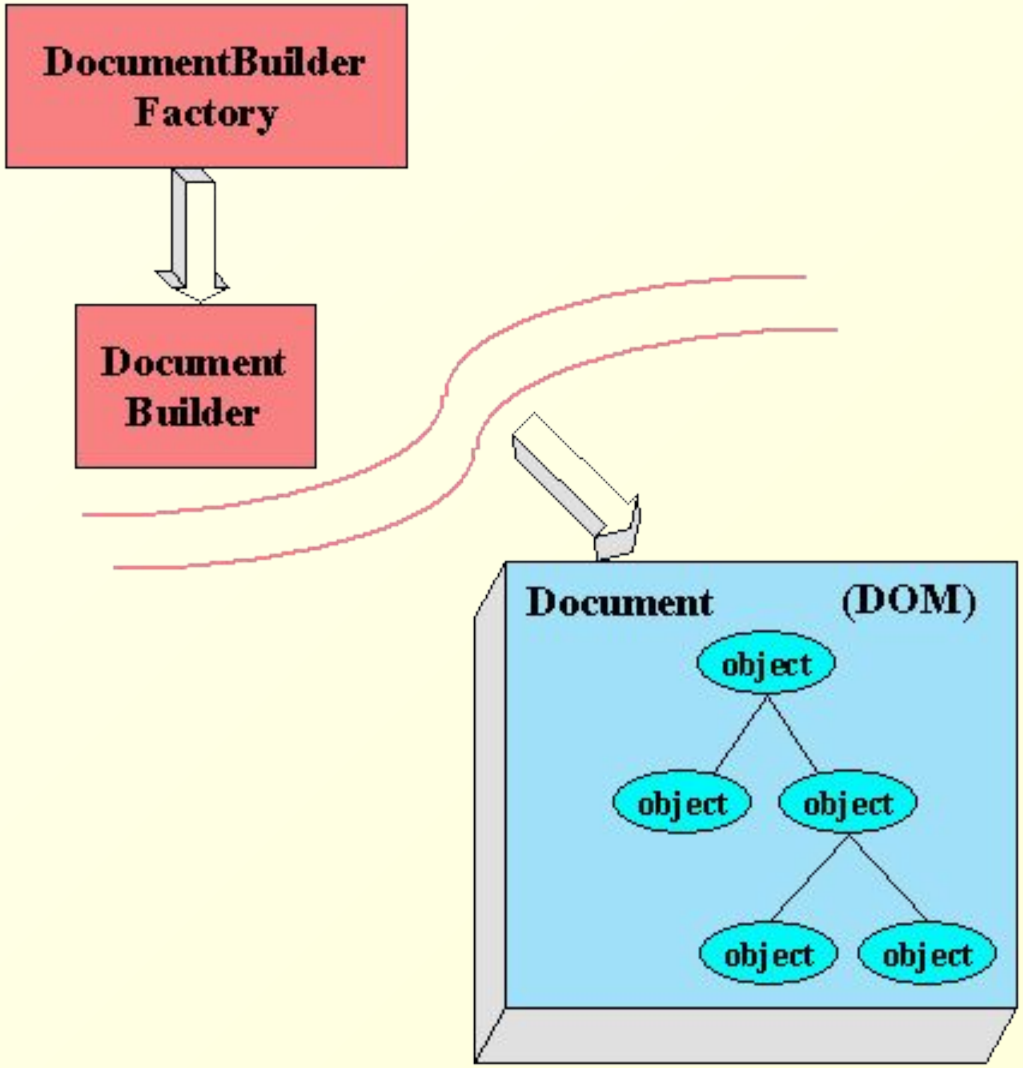
# The DOM Tree

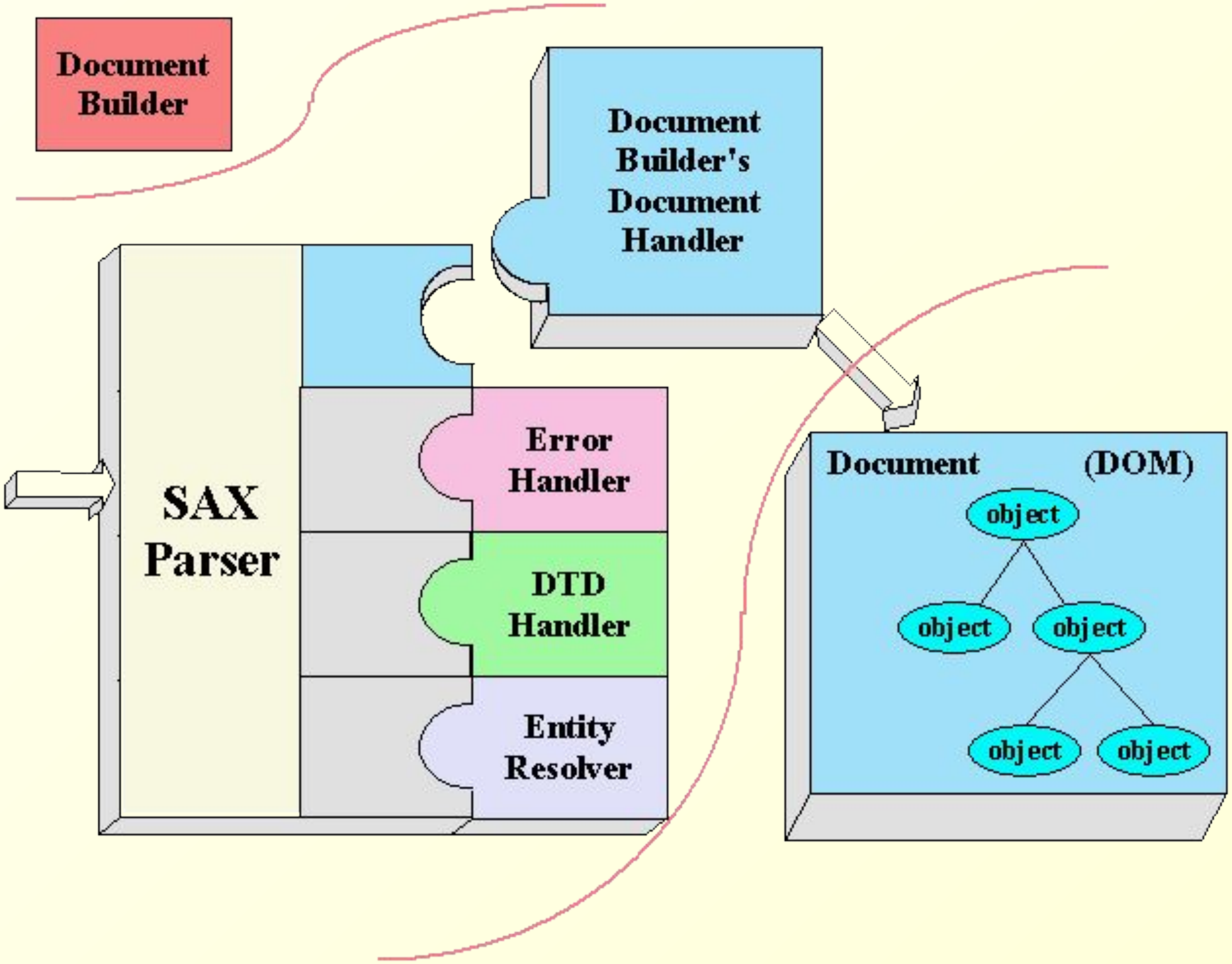


Document
countries
country
continent
Asia
name
Israel
population
year
2001
6,199,008
city
capital
yes
name
Jerusalem
city
capital
no
name
Ashdod
country
continent
Europe
name
France
population
year
2004
60,424,213

# Using a DOM Tree




DocumentBuilder Factory
Document Builder
Document
(DOM)
object
object
object
object
object


Document Builder
Document Builder's Document Handler
SAX Parser
Error Handler
DTD Handler
Entity Resolver
Document (DOM)
object
object
object
object
object

## Creating a DOM Tree

- A DOM tree is generated by a `DocumentBuilder`
- The builder is generated by a factory, in order to be implementation independent
- The factory is chosen according to the system configuration

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse("world.xml");
```

## Configuring the Factory

- The methods of the document-builder factory enable you to configure the properties of the document building
- For example
  - factory.setIgnoringElementContentWhitespace(true);
  - factory.setValidating(true)
  - factory.setIgnoringComments(false)

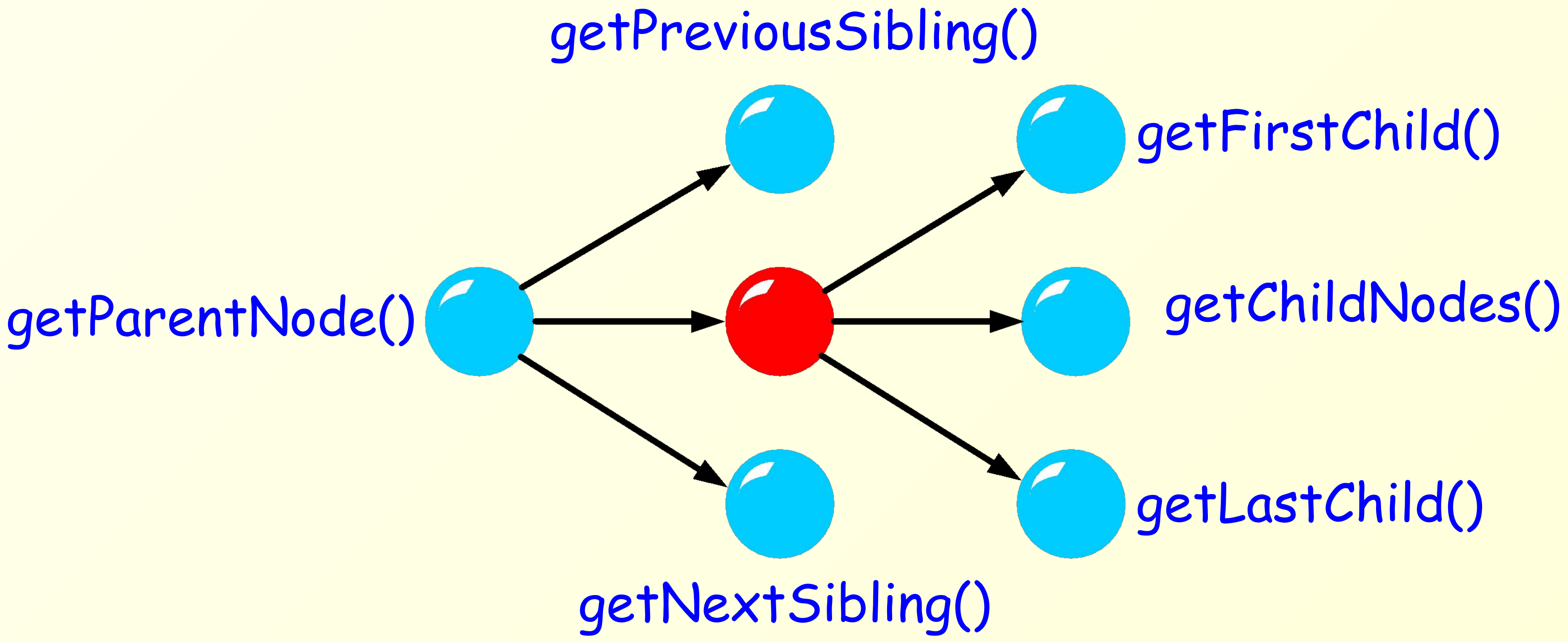
## The **`Node`** Interface

- The nodes of the DOM tree include
  - a special root (denoted *document*)
  - element nodes
  - text nodes and CDATA sections
  - attributes
  - comments
  - and more ...
- 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 (cont)

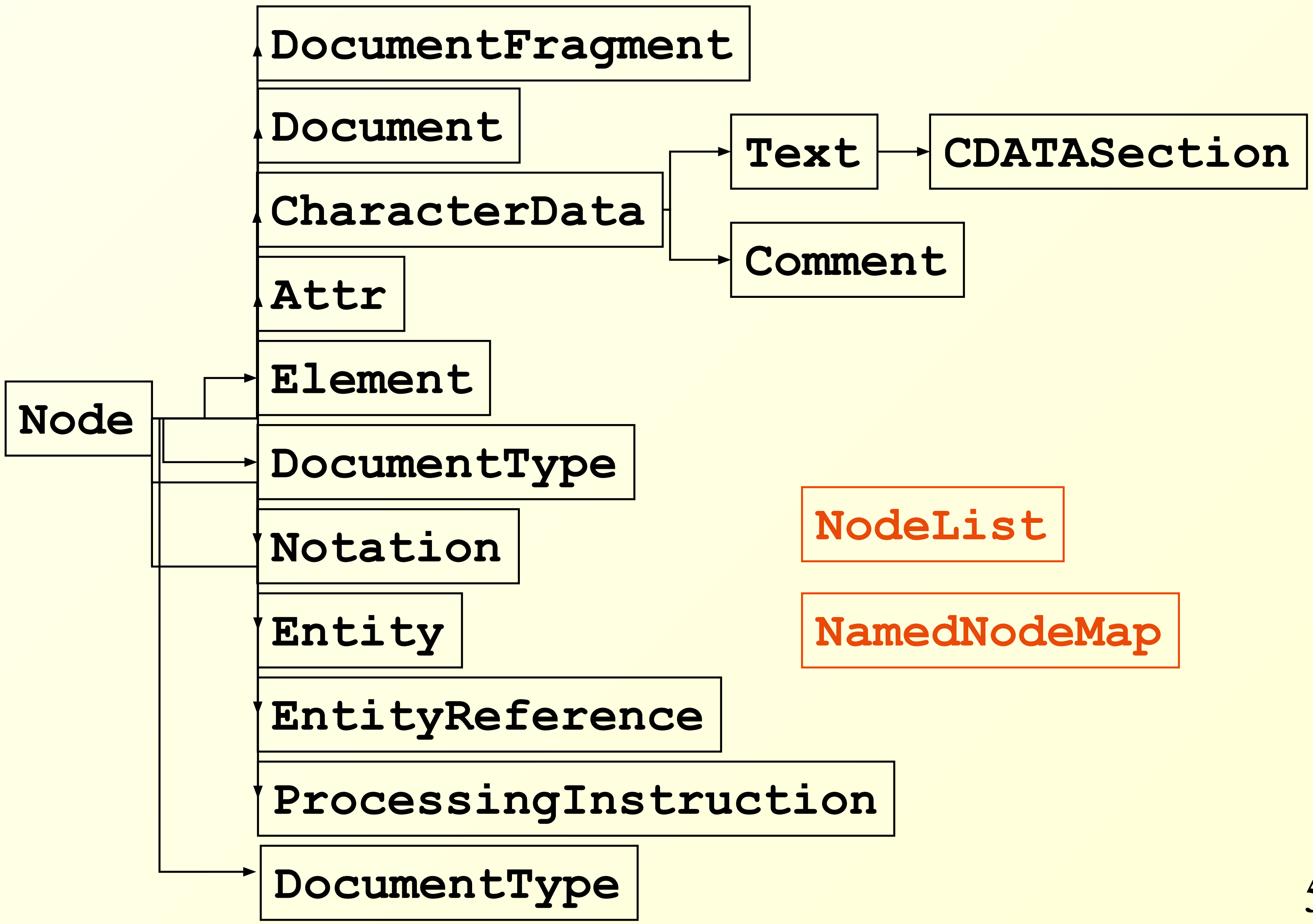


## Node Properties

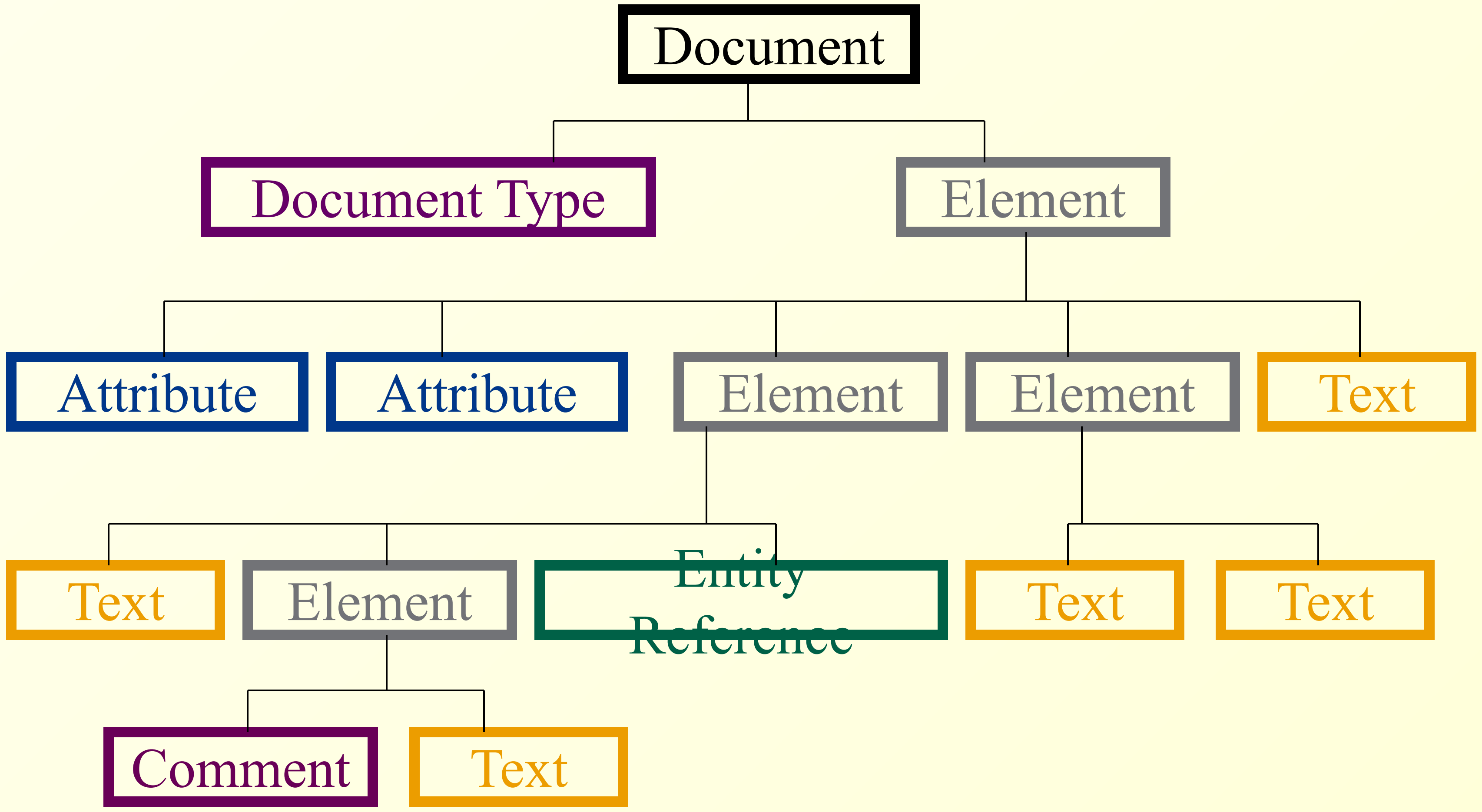
- Every node has
  - a type
  - a name
  - a value
  - attributes
- The roles of these properties differ according to the node types
- Nodes of different types implement different interfaces (that extend **`Node`**)

Figure as appears in : “The XML Companion” - Neil Bradley

# Interfaces in a DOM Tree



# Interfaces in the DOM Tree



# Names, Values and Attributes

| Interface | nodeName | nodeValue | attributes |
|---|---|---|---|
| **Attr** | name of attribute | value of attribute | null |
| **CDATASection** | `"#cdata-section"` | content of the Section | null |
| **Comment** | `"#comment"` | content of the comment | null |
| **Document** | `"#document"` | null | null |
| **DocumentFragment** | `"#document-fragment"` | null | null |
| **DocumentType** | doc-type name | null | null |
| **Element** | tag name | null | NodeMap |
| **Entity** | entity name | null | null |
| **EntityReference** | name of entity referenced | null | null |
| **Notation** | notation name | null | null |
| **ProcessingInstruction** | target | entire content | null |
| **Text** | `"#text"` | content of the text node | null |

## Node Types - getNodeType()

| | |
|---|---|
| ELEMENT_NODE = 1 | PROCESSING_INSTRUCTION_NODE = 7 |
| ATTRIBUTE_NODE = 2 | COMMENT_NODE = 8 |
| TEXT_NODE = 3 | DOCUMENT_NODE = 9 |
| CDATA_SECTION_NODE = 4 | DOCUMENT_TYPE_NODE = 10 |
| ENTITY_REFERENCE_NODE = 5 | DOCUMENT_FRAGMENT_NODE = 11 |
| ENTITY_NODE = 6 | NOTATION_NODE = 12 |

```
if (myNode.getNodeType() == Node.ELEMENT_NODE) {
  //process node
  …
}
```

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

public class EchoWithDom {

  public static void main(String[] args) throws Exception {
    DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
    factory.setIgnoringElementContentWhitespace(true);
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document doc = builder.parse("world.xml");
    new EchoWithDom().echo(doc);
  }
```

```
private void echo(Node n) {
   print(n);
   if (n.getNodeType() == Node.ELEMENT_NODE) {
      NamedNodeMap atts = n.getAttributes();
      ++depth;
      for (int i = 0; i < atts.getLength(); i++) echo(atts.item(i));
      --depth;  }

   depth++;
   for (Node child = n.getFirstChild(); child != null;
      child = child.getNextSibling()) echo(child);
   depth--;
}
```

```
private int depth = 0;
private String[] NODE_TYPES = {
   "", "ELEMENT",   "ATTRIBUTE", "TEXT", "CDATA",
   "ENTITY_REF", "ENTITY", "PROCESSING_INST",
   "COMMENT", "DOCUMENT", "DOCUMENT_TYPE",
   "DOCUMENT_FRAG", "NOTATION" };
private void print(Node n) {
   for (int i = 0; i < depth; i++) System.out.print("   ");

   System.out.print(NODE_TYPES[n.getNodeType()] + ":");
   System.out.print("Name: "+ n.getNodeName());
   System.out.print("  Value: "+ n.getNodeValue()+"\n");
}}
```

## Another Example

```
public class WorldParser {


  public static void main(String[] args) throws Exception {
    DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
    factory.setIgnoringElementContentWhitespace(true);
    DocumentBuilder builder =
      factory.newDocumentBuilder();
    Document doc = builder.parse("world.xml");
    printCities(doc);
}
```

## Another Example (cont)

```
public static void printCities(Document doc) {
   NodeList cities = doc.getElementsByTagName("city");
   for(int i=0; i<cities.getLength(); ++i) {
      printCity((Element)cities.item(i));
   }
}

public static void printCity(Element city) {
   Node nameNode =
      city.getElementsByTagName("name").item(0);
   String cName = nameNode.getFirstChild().getNodeValue();
   System.out.println("Found City: " + cName);
}
```

## Normalizing the DOM Tree

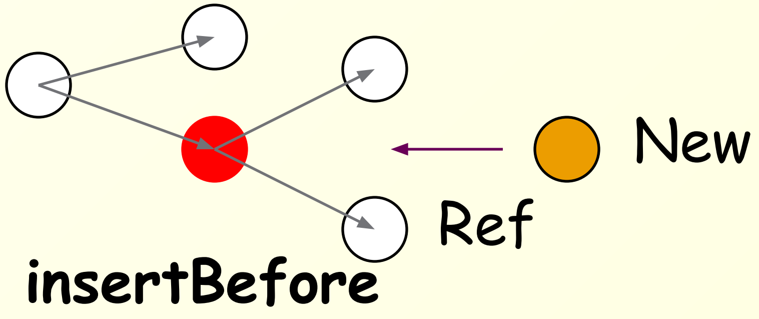
- Normalizing a DOM Tree has two effects:
  - Combine adjacent textual nodes
  - Eliminate empty textual nodes
- To normalize, apply the **`normalize()`** method to the document element

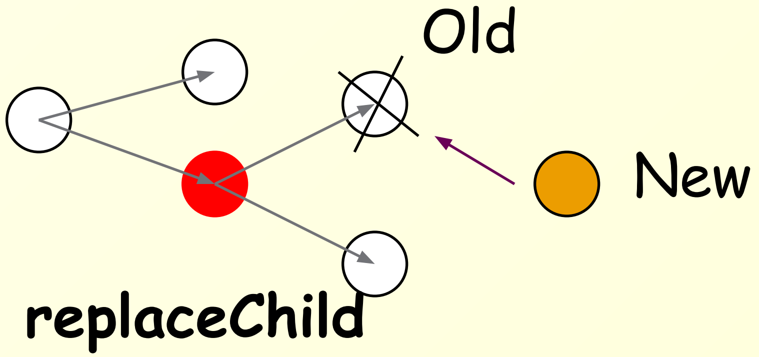
# Node Manipulation

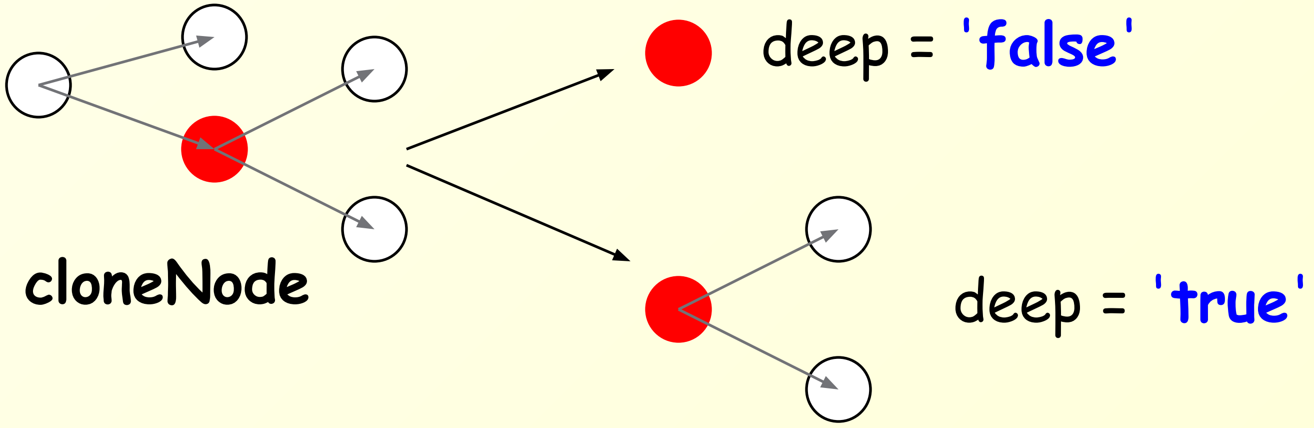
- Children of a node in a DOM tree can be manipulated - added, edited, deleted, moved, copied, etc.
- To constructs new nodes, use the methods of Document
  - createElement, createAttribute, createTextNode, createCDATASection etc.
- To manipulate a node, use the methods of Node
  - appendChild, insertBefore, removeChild, replaceChild, setNodeValue, cloneNode(boolean deep) etc.

Figure as appears in “The XML Companion” - Neil Bradley

# Node Manipulation (cont)







## SAX vs. DOM

## Parser Efficiency

- The DOM object built by DOM parsers is usually complicated and requires more memory storage than the XML file itself
  - A lot of time is spent on construction before use
  - For some very large documents, this may be impractical
- SAX parsers store only local information that is encountered during the serial traversal
- Hence, programming with SAX parsers is, in general, more efficient

## Programming using SAX is Difficult

- In some cases, programming with SAX is difficult:
  - How can we find, using a SAX parser, elements *e1* with ancestor *e2*?
  - How can we find, using a SAX parser, elements *e1* that have a descendant element *e2*?
  - How can we find the element *e1* referenced by the IDREF attribute of *e2*?

## Node Navigation

- SAX parsers do not provide access to elements other than the one currently visited in the serial (DFS) traversal of the document
- In particular,
  - They do not read backwards
  - They do not enable access to elements by ID or name
- DOM parsers enable any traversal method
- Hence, using DOM parsers is usually more comfortable

## More DOM Advantages

- DOM object ⇔ compiled XML
- You can save time and effort if you send and receive DOM objects instead of XML files
  - But, DOM object are generally larger than the source
- DOM parsers provide a natural integration of XML reading and manipulating
  - e.g., “cut and paste” of XML fragments

## Which should we use? DOM vs. SAX

- If your document is very large and you only need a few elements – use SAX
- If you need to manipulate (i.e., change) the XML – use DOM
- If you need to access the XML many times – use DOM (assuming the file is not too large)