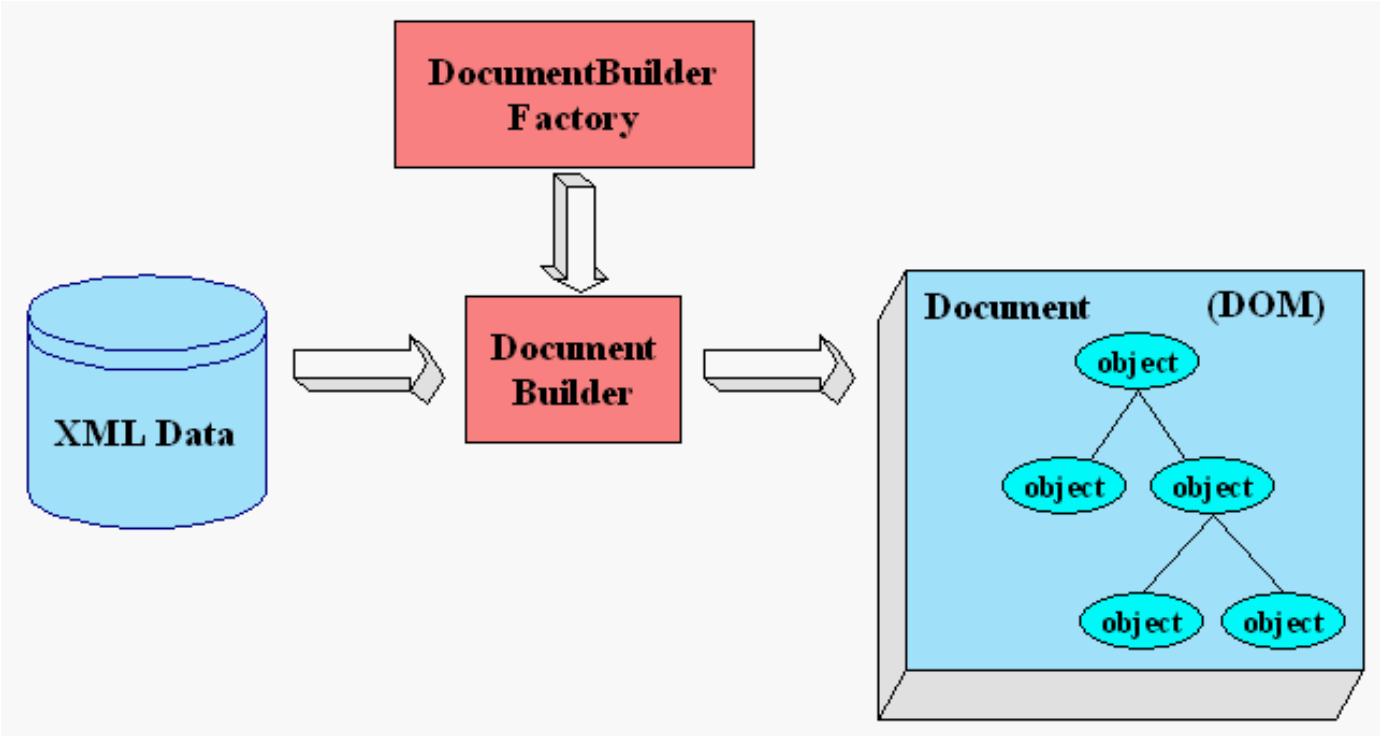
DOM architecture

DOM SAX

```
DocumentBuilderFactory dbf =  DocumentBuilderFactory.newInstance();
dbf.setValidating(true); // optional – default is non-validating
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(file);
```



DOM packages

DOM SAX

Package	*Description*
org.w3c.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers	Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. The factory that is used to create the builder is determined by the javax.xml.parsers system property, which can be set from the command line or overridden when invoking the newInstance method. This package also defines the ParserConfigurationException class for reporting errors.

The Node interface

DOM SAX

public interface Node

The Node interface is the primary datatype for the entire DOM. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a DOMException being raised.

The attributes nodeName, nodeValue and attributes are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (e.g., nodeValue for an Element or attributes for a Comment), this returns null. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

The Document interface

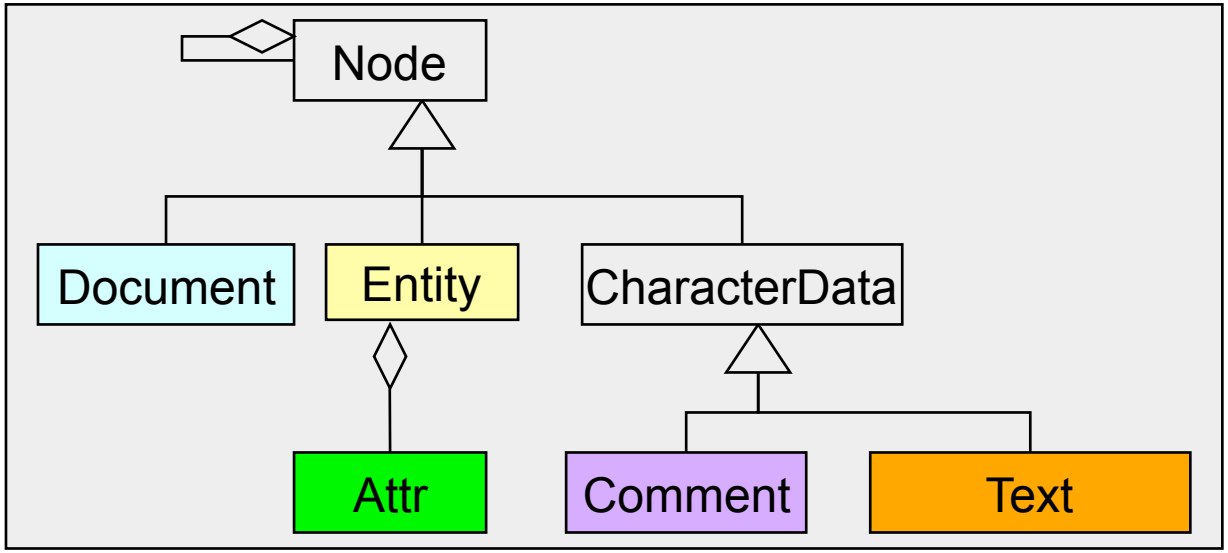
DOM SAX

public interface Document extends Node

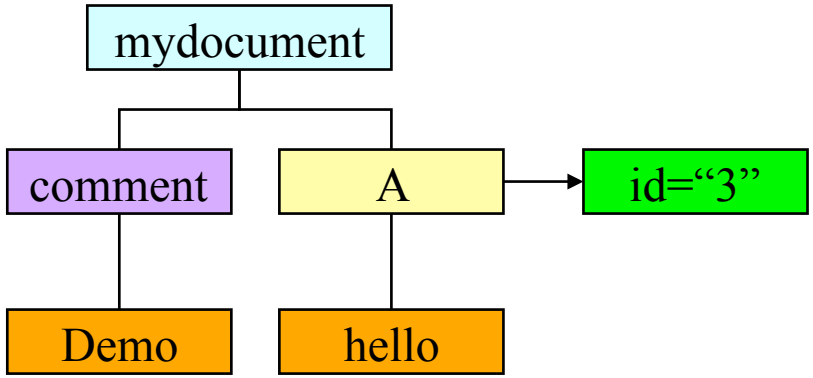
The **Document interface** represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data. Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

The Node hierarchy

DOM SAX

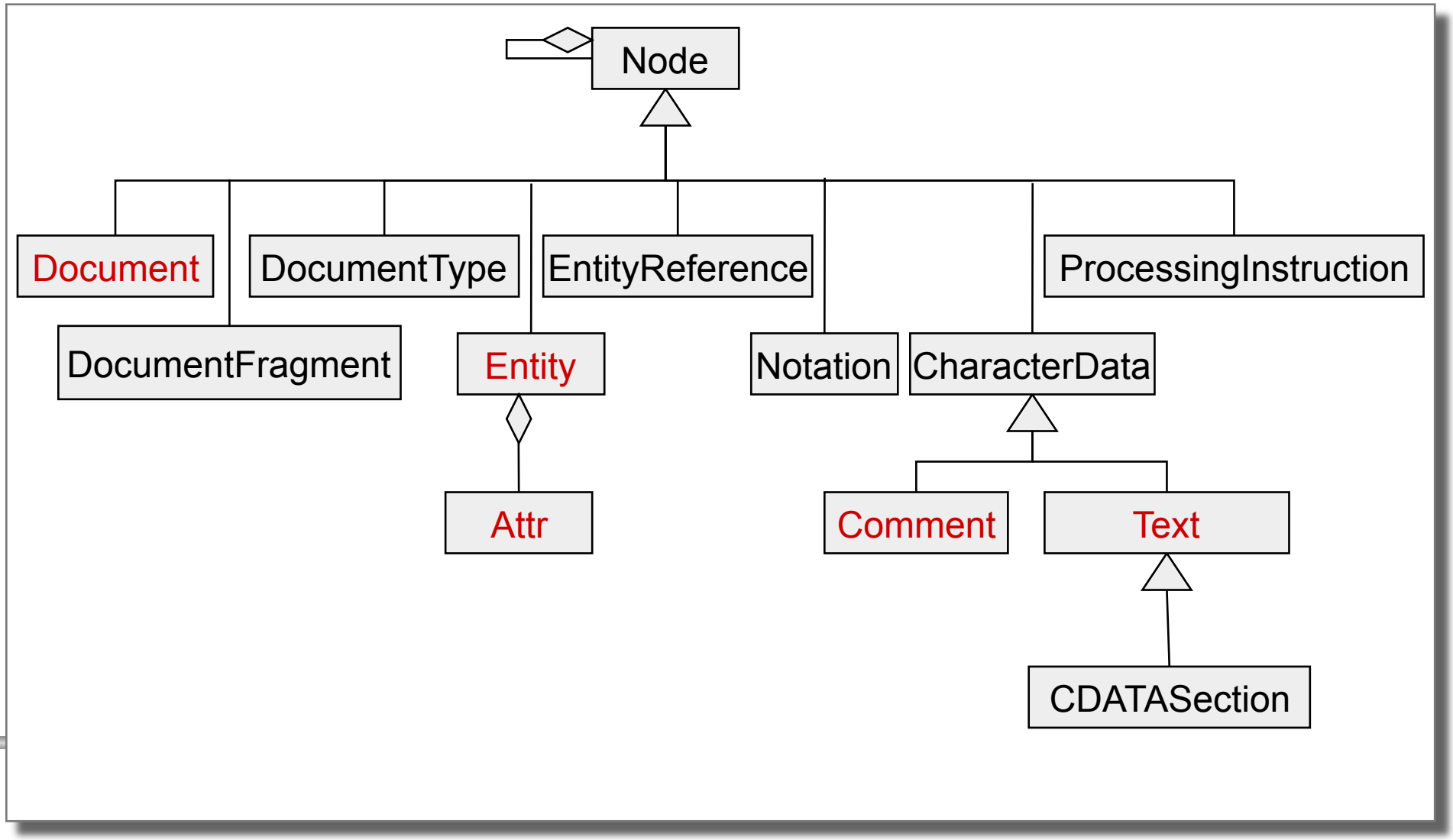



```
<!-- Demo -->
<A id="3">hello</A>
```



The Node hierarchy

DOM SAX



Node: WARNING!

DOM SAX

The implied semantic of this model is WRONG!

You might deduce that a comment might contain another comment, or a document, or any other node!

The integrity is delegated to a series of Node's attributes, that the programmer should check.

Node: main methods

DOM SAX

NAVIGATION

Node getParentNode()	The parent of this node.
NodeList getChildNodes()	A NodeList that contains all children of this node.
Node getFirstChild()	The first child of this node.
Node getLastChild()	The last child of this node.
Node getNextSibling()	The node immediately following this node
.	
Node getPreviousSibling()	The node immediately preceding this node.

The Node interface

DOM SAX

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

Node: main methods

DOM SAX

INSPECTION

java.lang.String getNodeName()
The name of this node, depending on its type; see table.

short getNodeType()
A code representing the type of the underlying object.

java.lang.String getNodeValue()
The value of this node, depending on its type; see the table.

Document getOwnerDocument()
The Document object associated with this node.

Boolean hasAttributes()
Returns whether this node (if it is an element) has any attributes.

Boolean hasChildNodes()
Returns whether this node has any children.

Node: main methods

DOM SAX

EDITING NODES

Node cloneNode(boolean deep)

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.

void setNodeValue(java.lang.String nodeValue)

The value of this node, depending on its type; see the table.

Node: main methods

DOM SAX

EDITING STRUCTURE

Node appendChild(Node newChild)

Adds the node newChild to the end of the list of children of this node.

Node removeChild(Node oldChild)

Removes the child node indicated by oldChild from the list of children, and returns it.

Node replaceChild(Node newChild, Node oldChild)

Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.

Node insertBefore(Node newChild, Node refChild)

Inserts the node newChild before the existing child node refChild.

void normalize()

Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.

NODE: determining the type

DOM SAX

```
switch (node.getNodeType()) {
        case Node.ELEMENT_NODE;  …; break;
        case Node.ATTRIBUTE_NODE;  …; break;
        case Node.TEXT_NODE;  …; break;
        case Node.CDATA_SECTION_NODE;  …; break;
        case Node.ENTITY_REFERENCE_NODE;  …; break;
        case Node.PROCESSING_INSTRUCTION;  …; break;
        case Node.COMMENT_NODE;  …; break;
        case Node.DOCUMENT_NODE;  …; break;
        case Node.DOCUMENT_TYPE_NODE;  …; break;
        case Node.DOCUMENT_FRAGMENT_NODE;  …; break;
        case Node.NOTATION_NODE;  …; break;
        default: throw (new Exception());
}
```

DOM example

DOM SAX

```
import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;   // parser uses SAX methods to build DOM object
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

public class CountDom {
  public static void main(String[ ] arg) throws Exception {
    if (arg.length != 1) {
      System.err.println("Usage: cmd filename (file must exist)");
      System.exit(1);
    }
     Node node = readFile(new File(arg[0]));
     System.out.println(arg + " elementCount: " + getElementCount(node));
    }
  }
```

DOM example

DOM SAX

```
public static Document readFile(File file) throws Exception {
    Document doc;
    try {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      dbf.setValidating(false);
      DocumentBuilder db = dbf.newDocumentBuilder();
      doc = db.parse(file);
      return doc;
    } catch (SAXParseException ex) {
      throw (ex);
    } catch (SAXException ex) {
      Exception x = ex.getException();   // get underlying Exception
      throw ((x == null) ? ex : x);
    }
  }
```

Parse File,
Return Document

DOM example

DOM SAX

```
public static int getElementCount(Node node) {
  if (null == node)      return 0;
  int sum = 0;
  boolean isElement = (node.getNodeType() == Node.ELEMENT_NODE);
  if (isElement)  sum = 1;
  NodeList children = node.getChildNodes();
  if (null == children)      return sum;

  for (int i = 0; i < children.getLength(); i++) {
    sum += getElementCount(children.item(i));   // recursive call
  }
  return sum;
 }
}
```

use DOM methods to count elements:
for each subtree if the root is an Element,
 set sum to 1, else to 0;
 add element count of all children of the root to sum

DOM References

DOM SAX

http://docs.oracle.com/javase/tutorial/jaxp/dom/index.html

Alternatives to DOM

"Build a better mousetrap, and the world will beat a path to your door."
--Emerson

Alternatives to DOM

If you are dealing with simple data structures and if XML Schema isn't a big part of your plans, then you may find that one of the more object-oriented standards, such as JDOM and dom4j, is better suited for your purpose.

JDOM: Java DOM (see http://www.jdom.org).

The standard DOM is a very simple data structure that intermixes text nodes, element nodes, processing instruction nodes, CDATA nodes, entity references, and several other kinds of nodes. That makes it difficult to work with in practice, because you are always sifting through collections of nodes, discarding the ones you don't need into order to process the ones you are interested in. JDOM, on the other hand, creates a tree of *objects* from an XML structure. The resulting tree is much easier to use, and it can be created from an XML structure without a compilation step.

DOM4J: DOM for Java (see http://www.dom4j.org/)

***dom4j* is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP.**