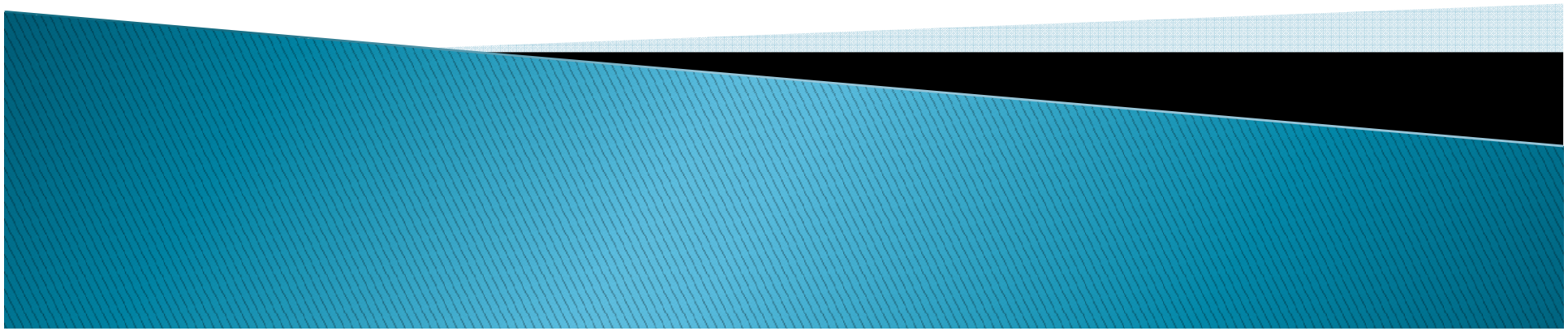
XML
DOM and SAX Parsers

Reference

"XML a Manager's Guide" by Kevin Dick

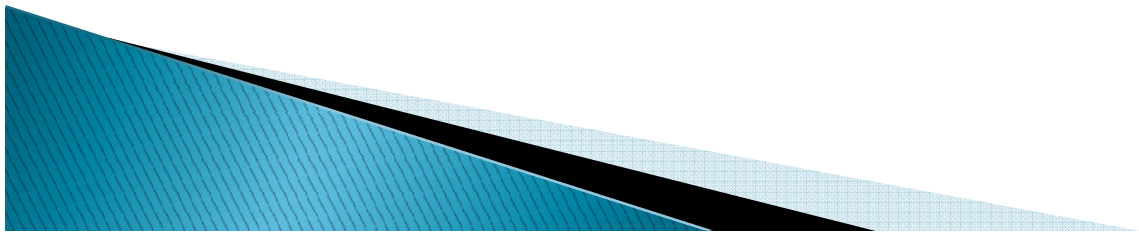
"The XML Companion" by Bradley

http://www.w3schools.com/xml/dom_intro.asp



Introduction to parsers

- The word *parser* comes from compilers
- In a compiler, a parser is the module that reads and interprets the programming language.



Introduction to Parsers

- In XML, a parser is a software component that sits between the application and the XML files.

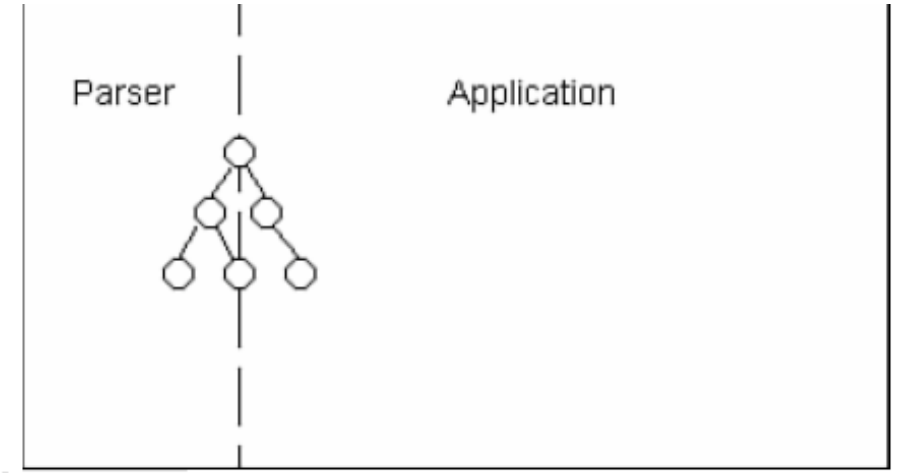
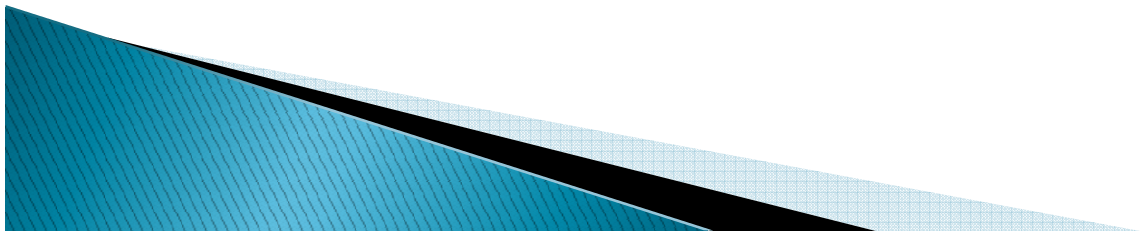


Figure 1: architecture of an xml program

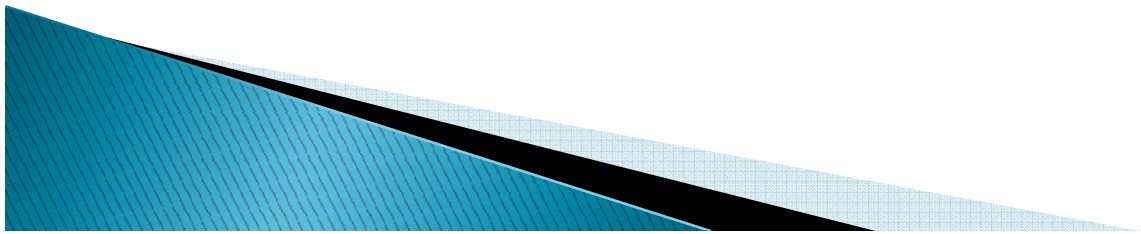
Introduction to parsers

- It reads a text-formatted XML file or stream and converts it to a document to be manipulated by the application.



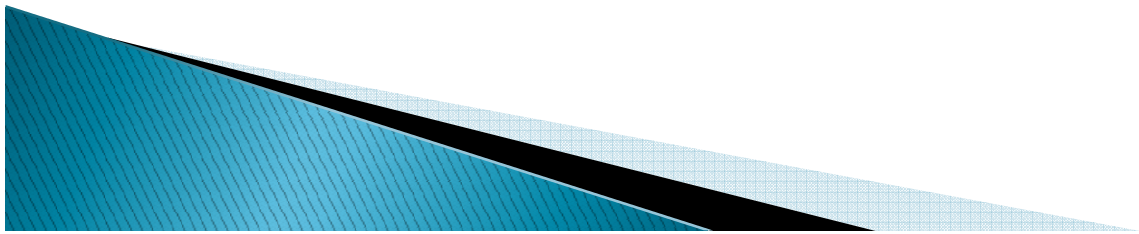
Well-formedness and validity

- Well-formed documents respect the syntactic rules.
- Valid documents not only respect the syntactic rules but also conform to a structure as described in a DTD.



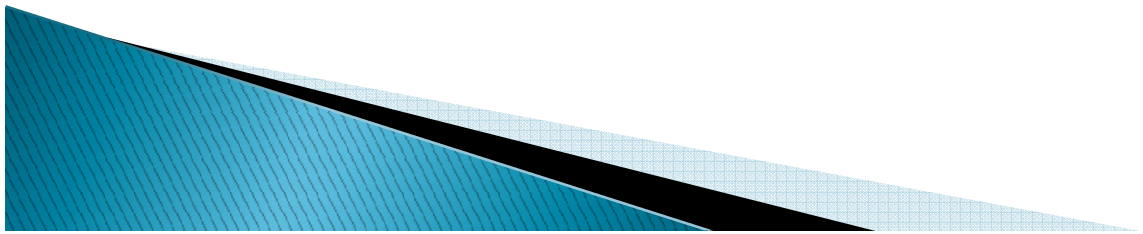
Validating vs. Non-validating parsers

- Both parsers enforce syntactic rules
- only validating parsers know how to validate documents against their DTDs



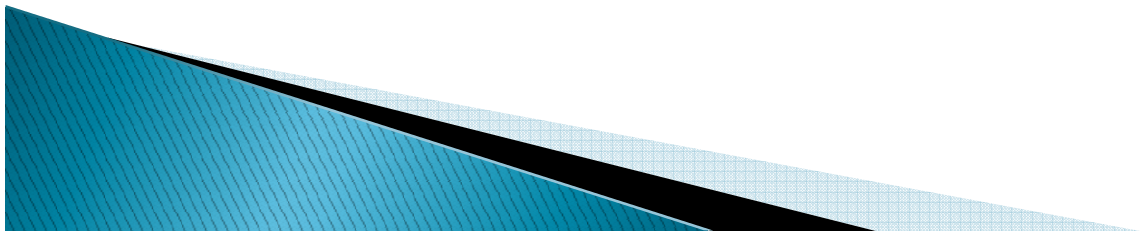
Tree-based parsers

- These map an XML document into an internal tree structure, and then allow an application to navigate that tree.
- Ideal for browsers, editors, XSL processors.



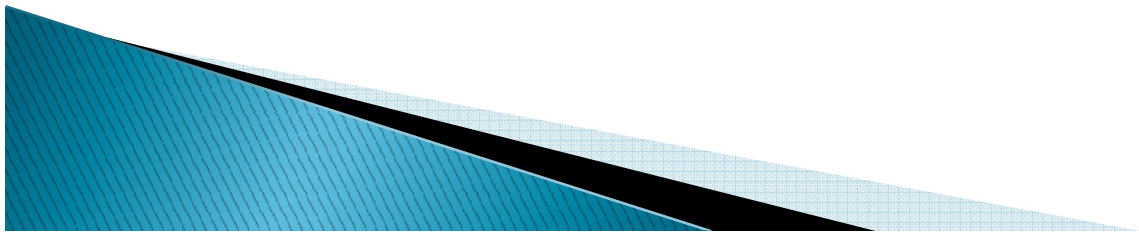
Event-based

- An event-based API reports parsing events (such as the start and end of elements) directly to the application through callbacks.
- The application implements handlers to deal with the different events



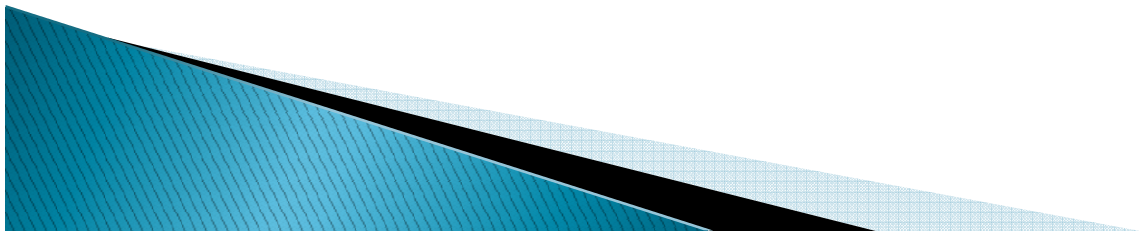
Event-based vs. Tree-based parsers

- Tree-based parsers deal generally small documents.
- Event-based parsers deal generally used for large documents.



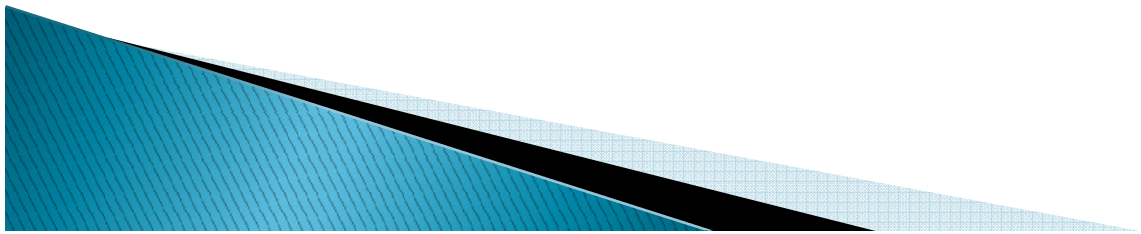
Event-based vs. Tree-based parsers

- Tree-based parsers are generally easier to implement.
- Event-based parsers are more complex and give hard time for the programmer



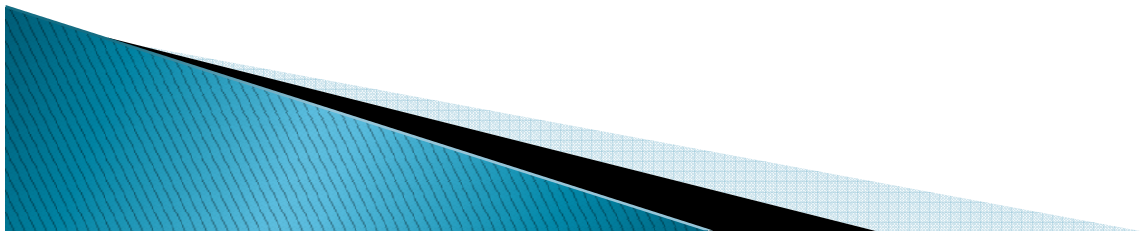
What is DOM?

- The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents.
- It defines the logical structure of documents and the way a document is accessed and manipulated



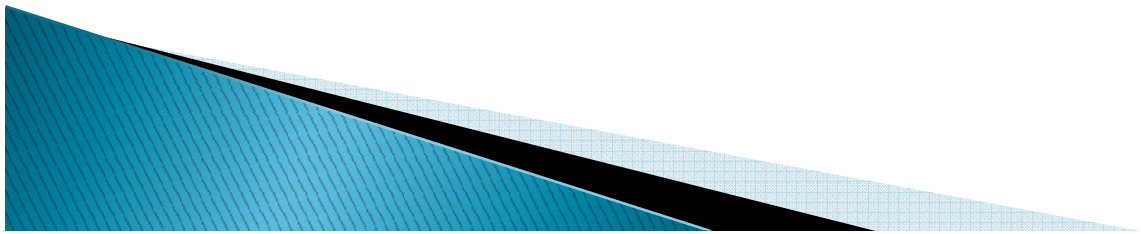
Properties of DOM

- Programmers can build documents, navigate their structure, and add, modify, or delete elements and content.
- Provides a standard programming interface that can be used in a wide variety of environments and applications.
- structural isomorphism.



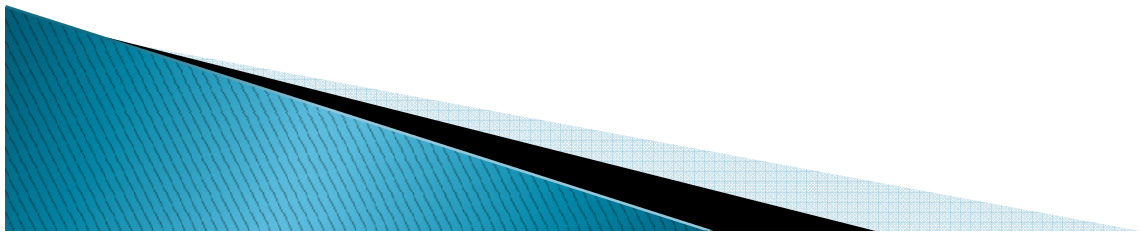
DOM Identifies

- The interfaces and objects used to represent and manipulate a document.
- The semantics of these interfaces and objects – including both behavior and attributes.
- The relationships and collaborations among these interfaces and objects.



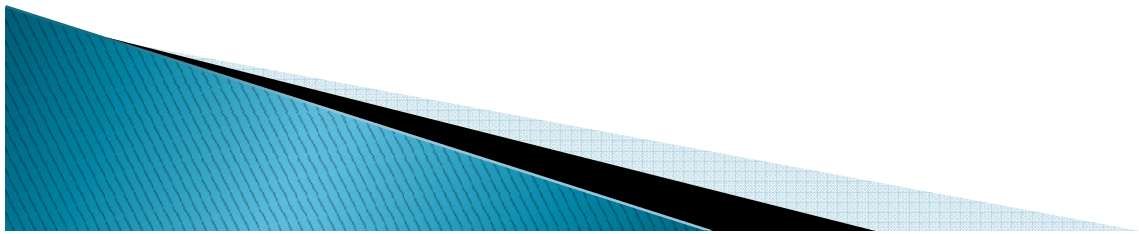
What DOM is not!!

- The Document Object Model is not a binary specification.
- The Document Object Model is not a way of persisting objects to XML or HTML.
- The Document Object Model does not define "the true inner semantics" of XML or HTML.



What DOM is not!!

- The Document Object Model is not a set of data structures, it is an object model that specifies interfaces.
- The Document Object Model is not a competitor to the Component Object Model (COM).



DOM into work

```
<?xml version="1.0"?>
<products>
   <product>
         <name>XML Editor</name>
         <price>499.00</price>
   </product>
   <product>
         <name>DTD Editor</name>
         <price>199.00</price>
   </product>
   <product>
         <name>XML Book</name>
         <price>19.99</price>
   </product>
   <product>
         <name>XML Training</name>
         <price>699.00</price>
   </product>
</products>
```

DOM into work

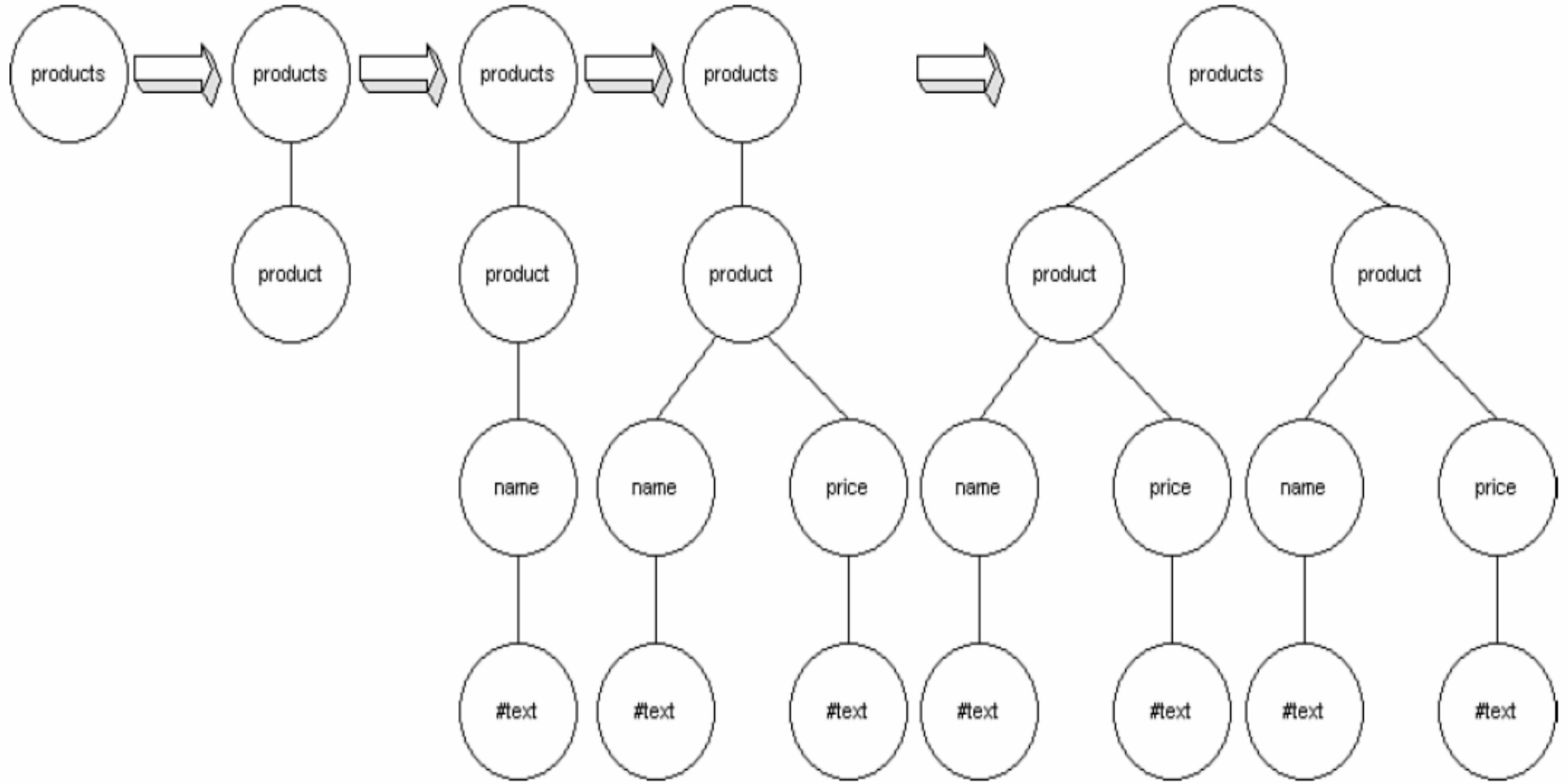
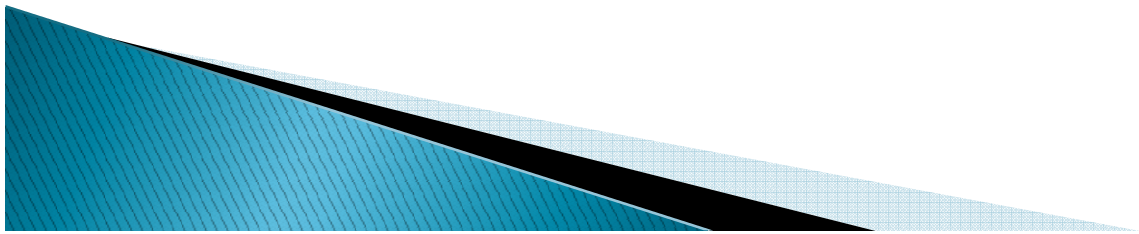


Figure 1: Building the tree of objects

DOM levels: level 0

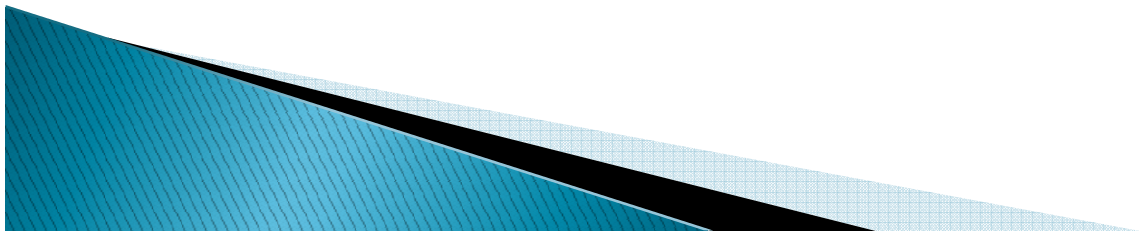
- DOM Level 0 is a mix of Netscape Navigator 3.0 and MS Internet Explorer 3.0 document functionalities.



DOM levels: DOM 1

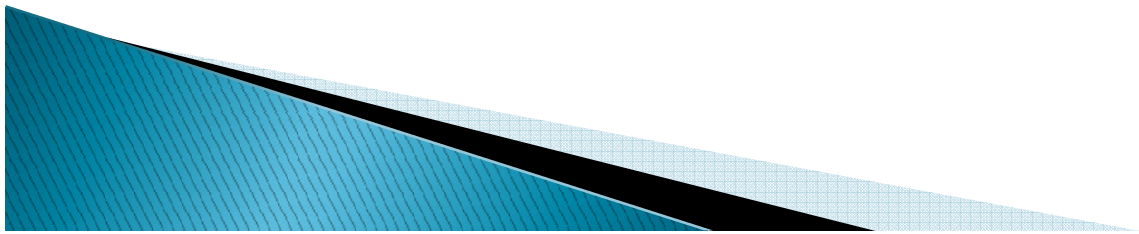
- It contains functionality for document navigation and manipulation.

i.e.: functions for creating, deleting and changing elements and their attributes.



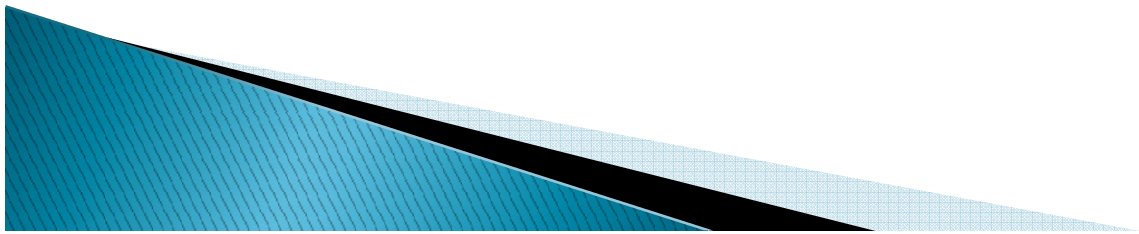
DOM level 1 limitations

- A structure model for the internal subset and the external subset.
- Validation against a schema.
- Control for rendering documents via style sheets.
- Access control.
- Thread-safety.
- Events



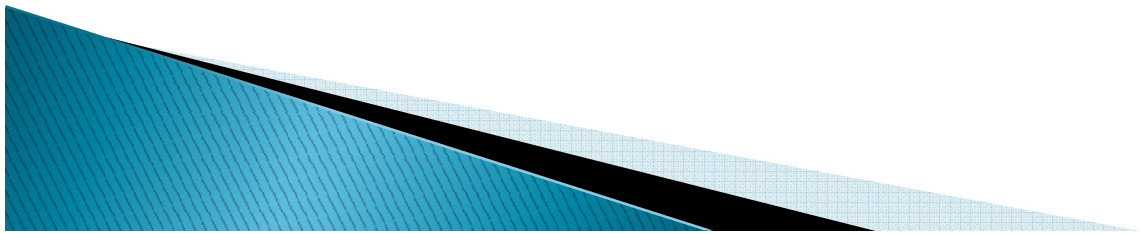
DOM levels: DOM 2

- A style sheet object model and defines functionality for manipulating the style information attached to a document.
- Enables traversal on the document.
- Defines an event model.
- Provides support for XML namespaces



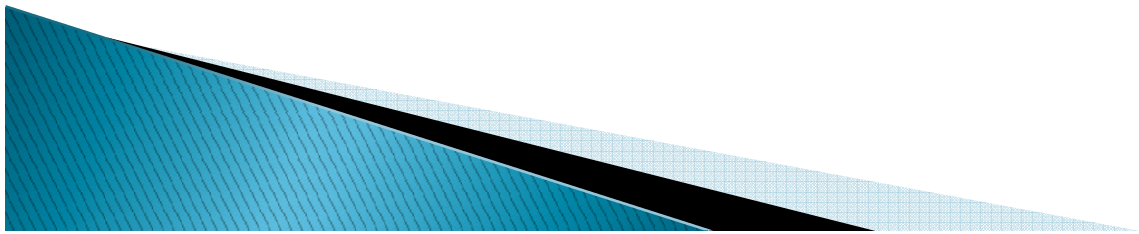
DOM levels: DOM 3

- Document loading and saving as well as content models (such as DTD's and schemas) with document validation support.
- Document views and formatting, key events and event groups



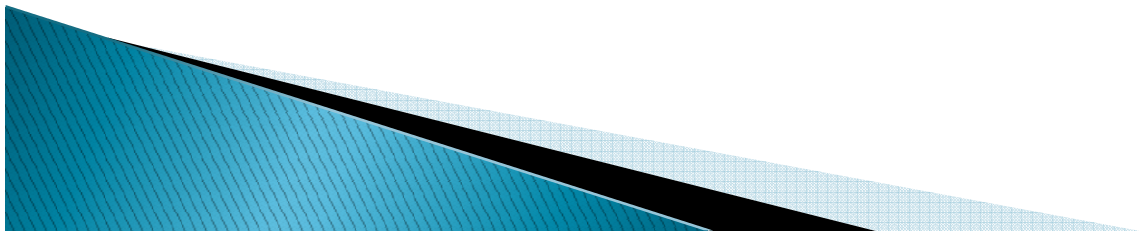
Processing XML with DOM

- The following examples were tested using Sun's JAXP (Java API for XMP Parsing. This is available at http://www.javasoft.com/ and click on XML



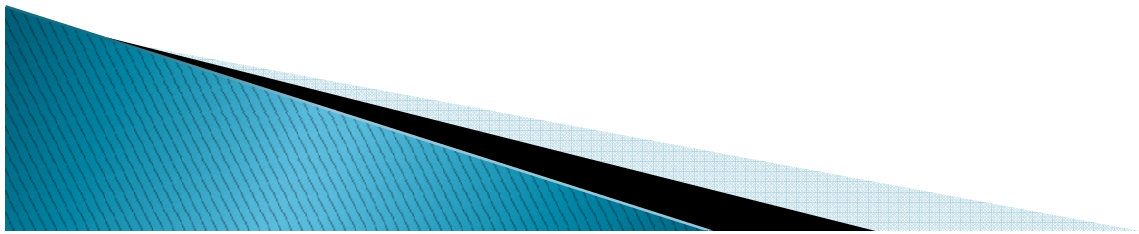
XML DOM

- The World Wide Web Consortium's Document Object Model
- Provides a common vocabulary to use in manipulating XML documents.
- May be used from C, Java, Perl, Python, or VB
- Things may be quite different "under the hood".
- The interface to the document will be the same.

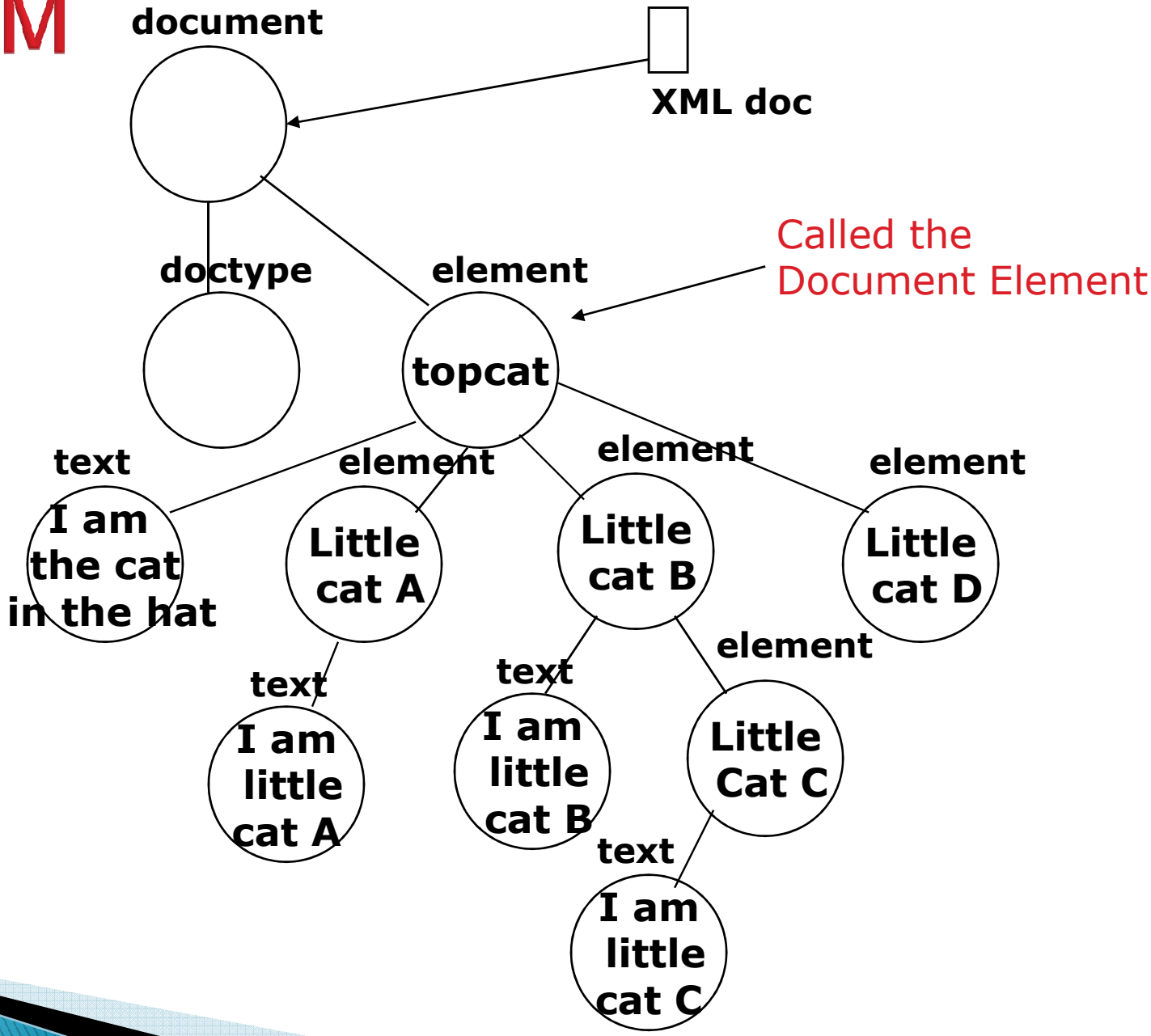


XML DOM

```
<?xml version = "1.0" ?>
<!DOCTYPE TopCat SYSTEM "cats.dtd">
<TopCat>
      I am The Cat in The Hat
      <LittleCatA>
          I am Little Cat A
      </LittleCatA>
     <LittleCatB>
          I am Little Cat B
          <LittleCatC>
              I am Little Cat C
          </LittleCatC>
     </LittleCatB>
  <LittleCatD/>
</TopCat>
```



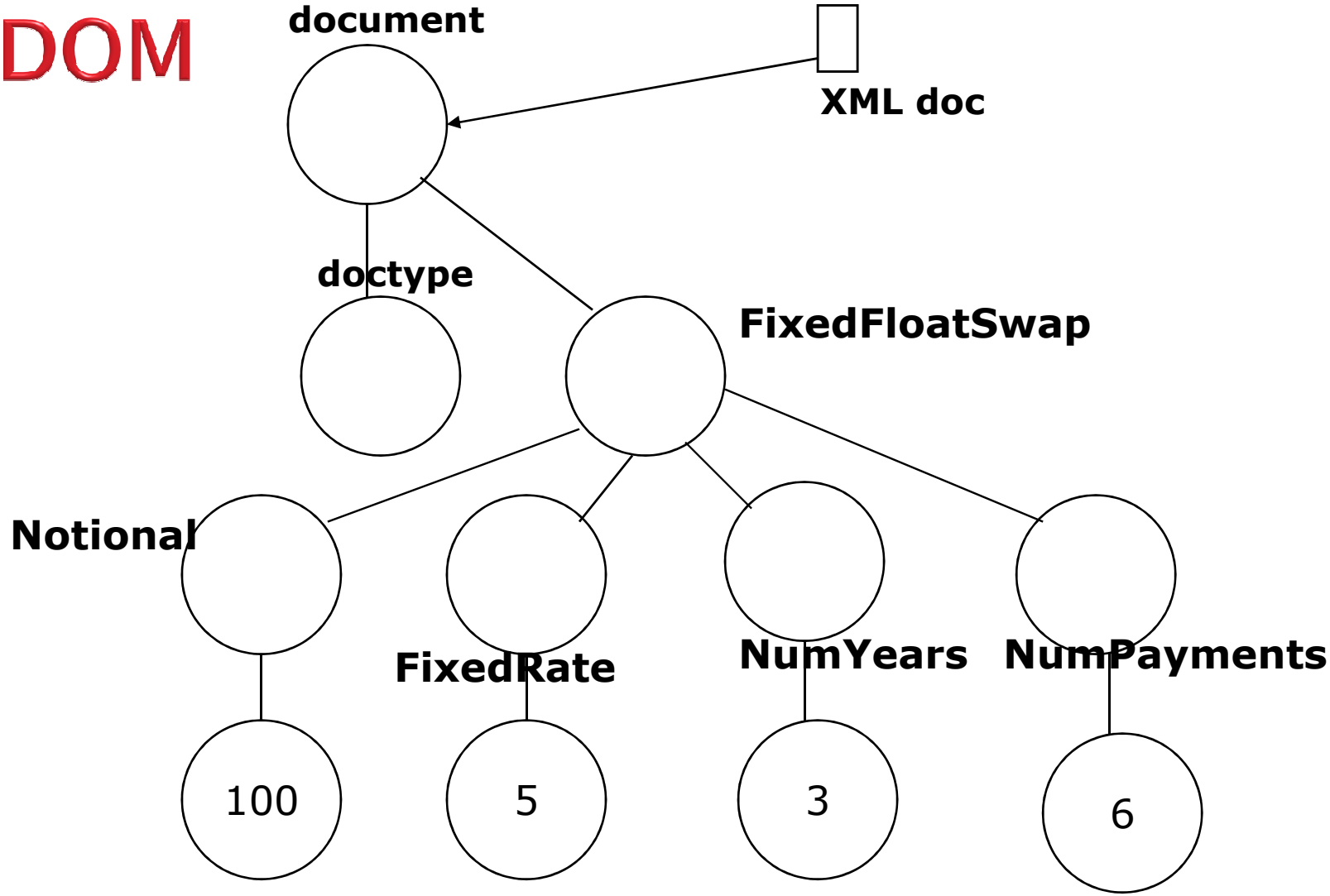
XML DOM


document
XML doc
doctype
element
Called the
Document Element
topcat
text
element
element
element
I am
the cat
in the hat
Little
cat A
Little
cat B
Little
cat D
text
text
element
I am
little
cat A
I am
little
cat B
Little
Cat C
text
I am
little
cat C

Agreement.xml

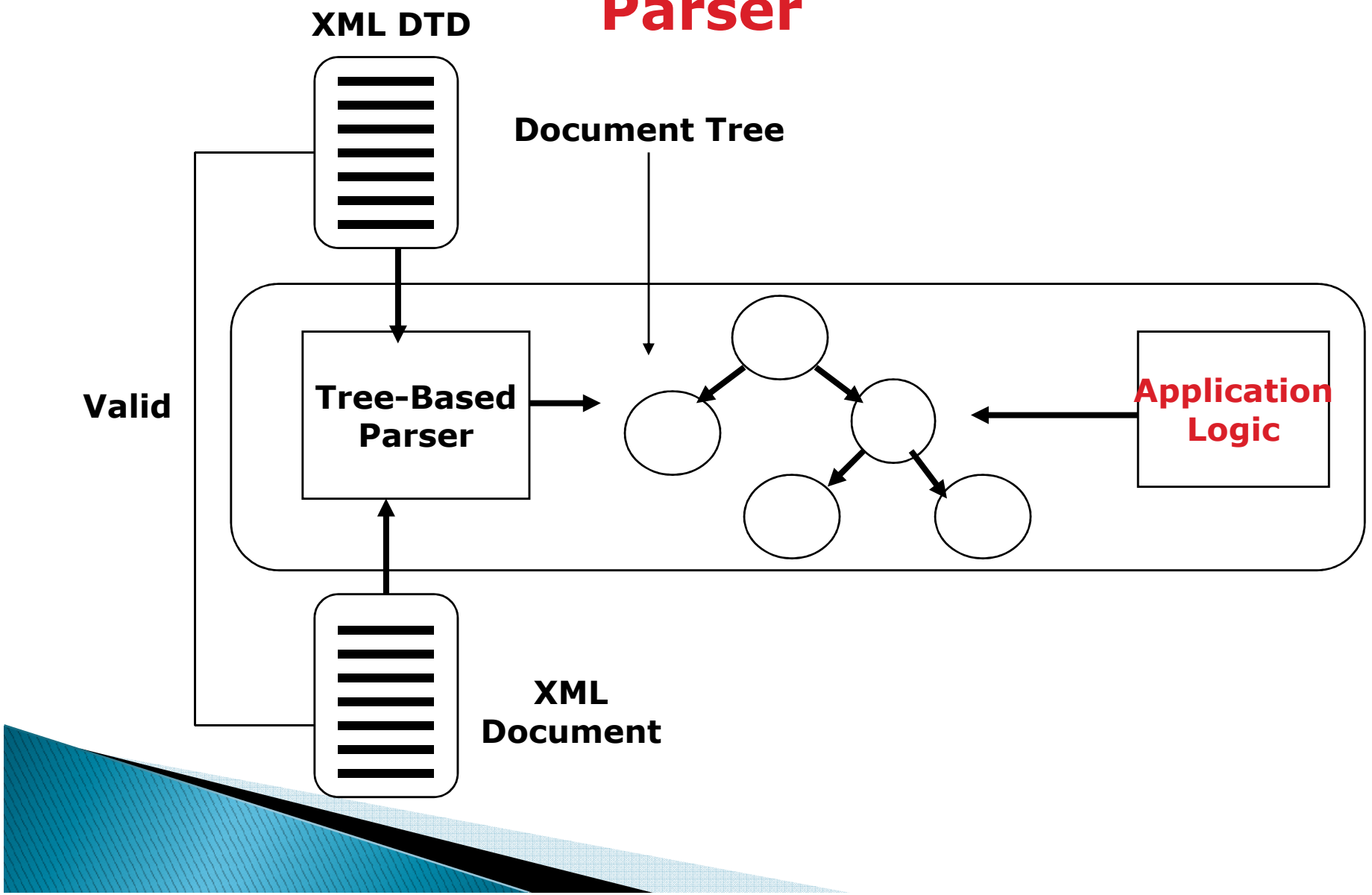
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE FixedFloatSwap SYSTEM "FixedFloatSwap.dtd">
<FixedFloatSwap>
    <Notional>100</Notional>
    <Fixed_Rate>5</Fixed_Rate>
    <NumYears>3</NumYears>
    <NumPayments>6</NumPayments>
</FixedFloatSwap>
```

XML DOM



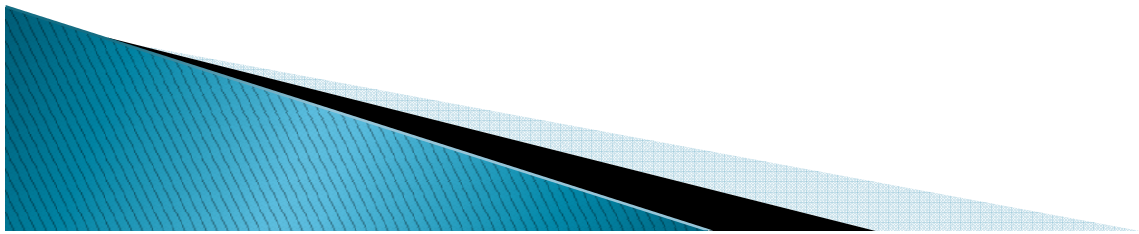
All of these nodes implement the Node interface

Operation of a Tree-based Parser



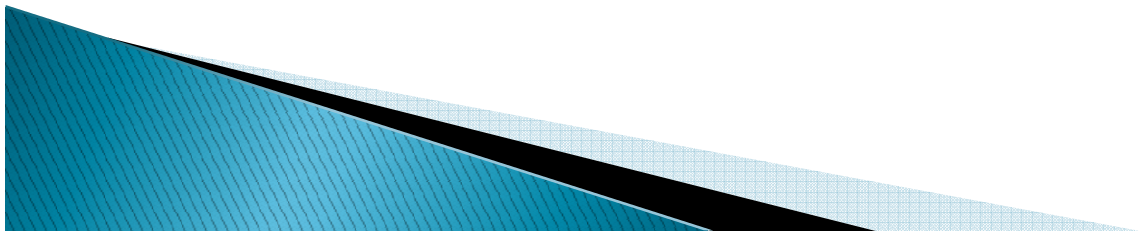
The Node Interface

- The Node interface is the primary datatype for the entire Document Object Model.
- 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.



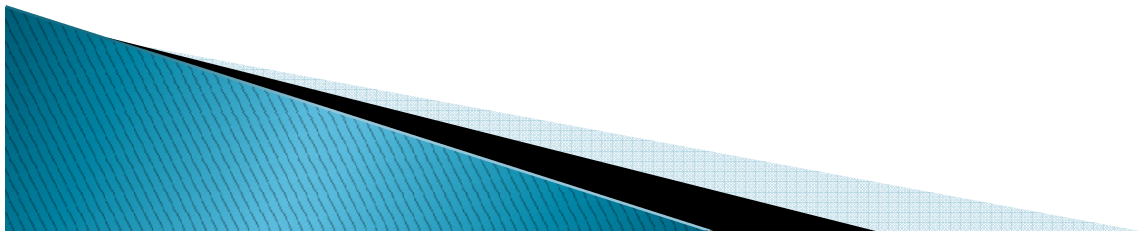
The Node Interface

- 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

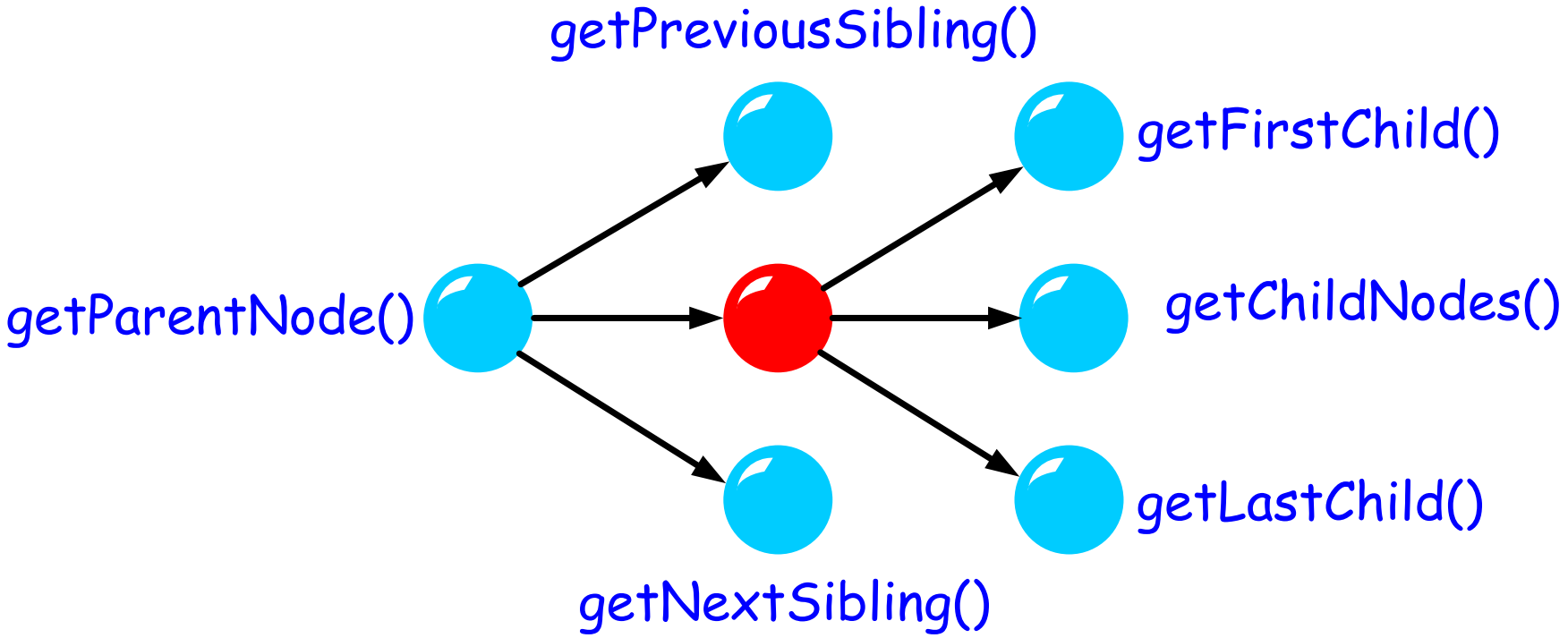


Node Navigation

- 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()`

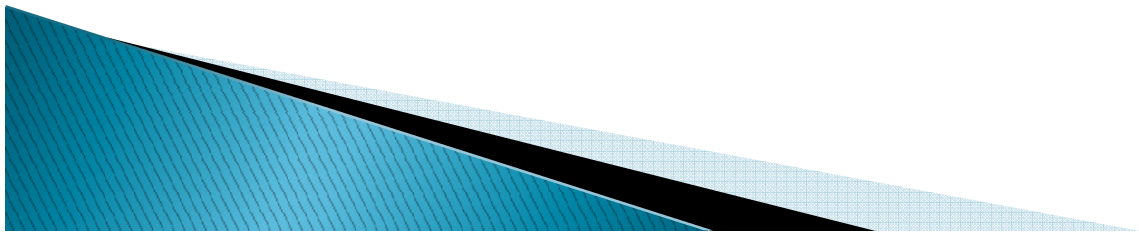


Node Navigation (cont)



Properties

- All Nodes have properties.
- Not all properties are needed by all types of nodes.
- The attribute property is an important part of the Element node but is null for the Text nodes.
- We access the properties through methods…



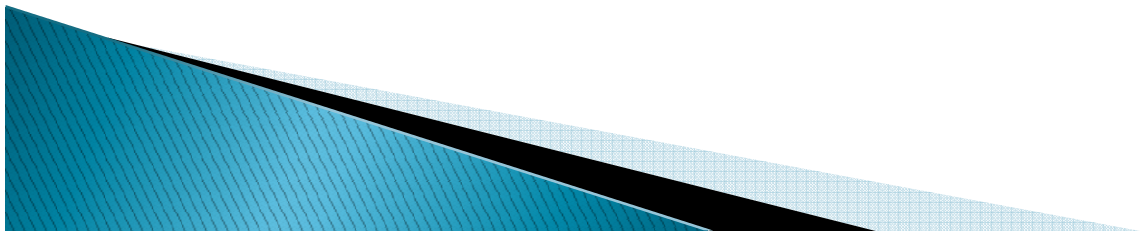
Some Methods of Node

Example Methods are:

String getNodeName() – depends on the Node type

if Element node return tag name

if Text node return #text

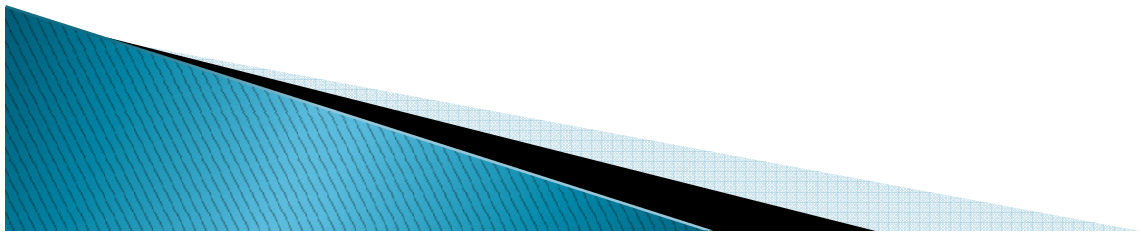


Some Methods of Node

Example Methods are:

short getNodeType()

Might return a constant like ELEMENT_NODE or TEXT_NODE or …



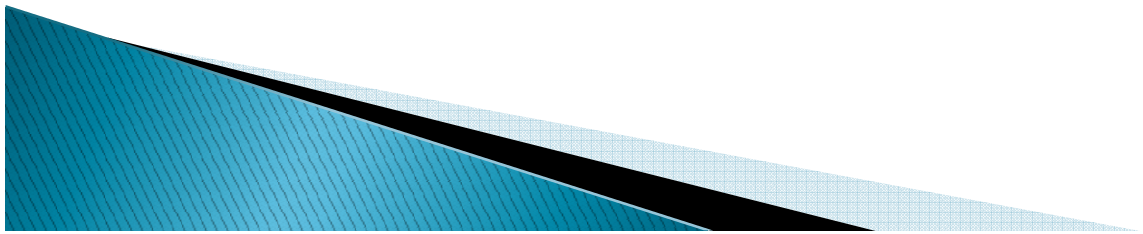
Some Methods of Node

Example Methods are:

String getNodeValue()

if the Node is an Element Node then return ‘null’

if the Node is a Text Node then return a String representing that text.

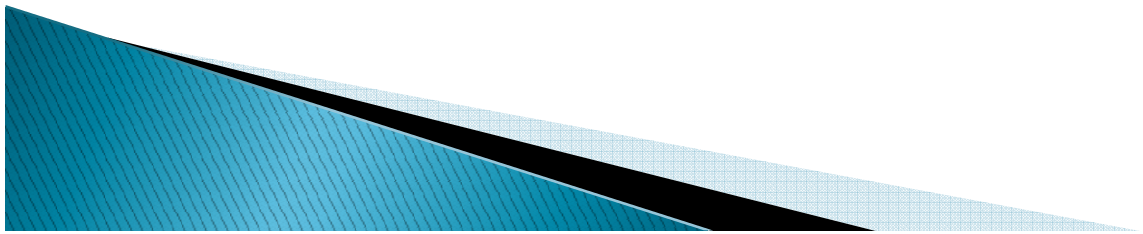


Some Methods of Node

Example Methods are:

Node getParentNode()

returns a reference to the parent

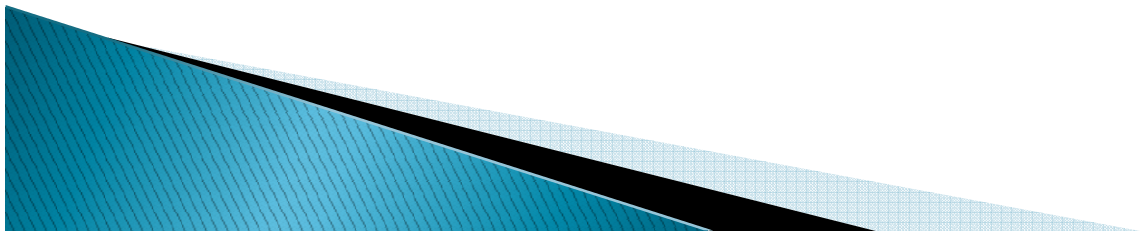


Some Methods of Node

Example Methods are:

public Node getFirstChild()

Returns the value of the firstChild property.



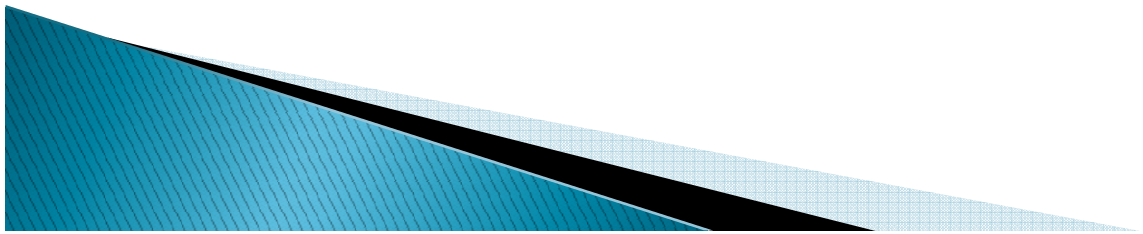
Some Methods of Node

Example Methods are:

public NodeList getChildNodes()

returns a NodeList object

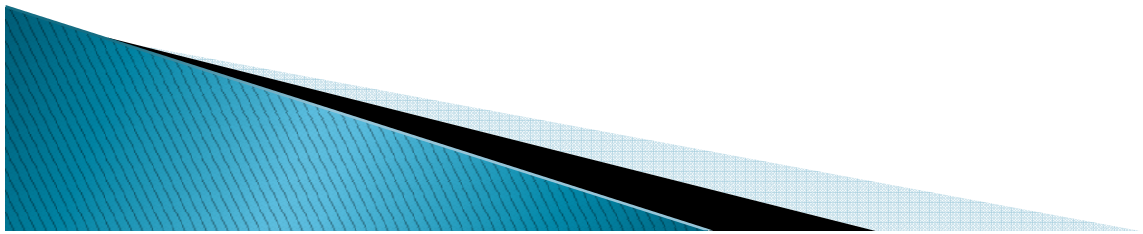
NodeList is an interface and not a Node.



The NodeList Interface

- The NodeList interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.

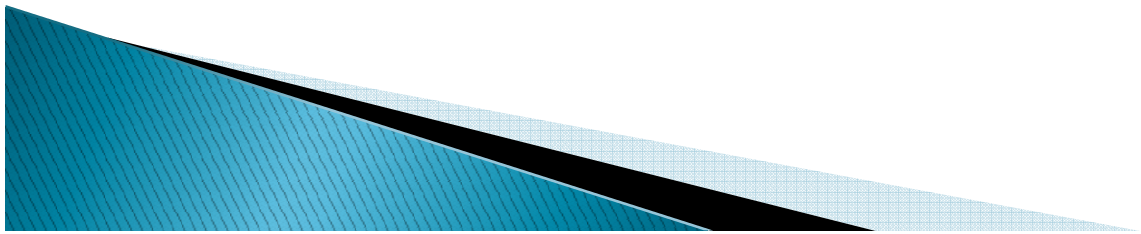
- The items in the NodeList are accessible via an integral index, starting from 0.



Methods of NodeList Interface

public Node item(int index)

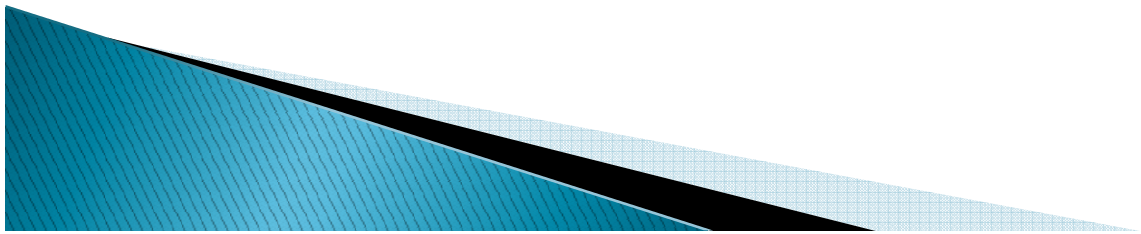
Returns the item at index in the collection. If index is greater than or equal to the number of nodes in the list, this returns null.



Methods of NodeList Interface

public int getLength()

Returns the value of the length property.



The Element Interface

public interface Element
extends Node

Inheritance

- By far the vast majority of objects (apart from text) that authors encounter when traversing a document are Element nodes.

Nothing prevents us from extending one interface in order to create another.

- Those who implement Element just have more promises to keep.

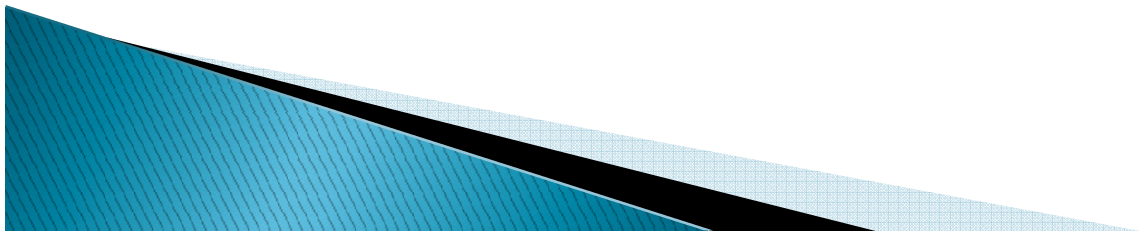
The Element Interface

public interface Element
extends Node

- Some methods in the Element interface

String getAttribute(String name)

Retrieves an attribute value by name.



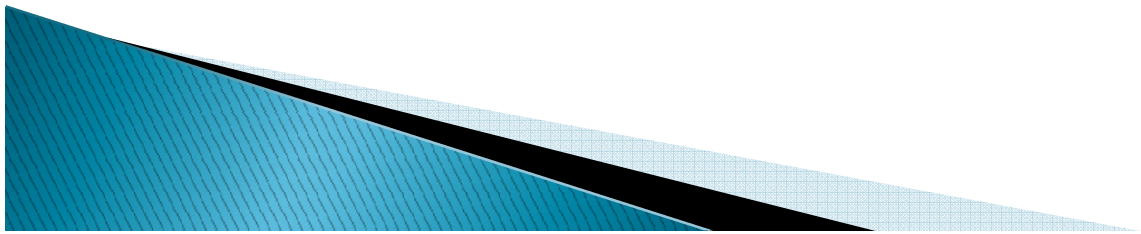
The Element Interface

public interface Element
extends Node

- Some methods in the Element interface

public String getTagName()

Returns the value of the tagName property.



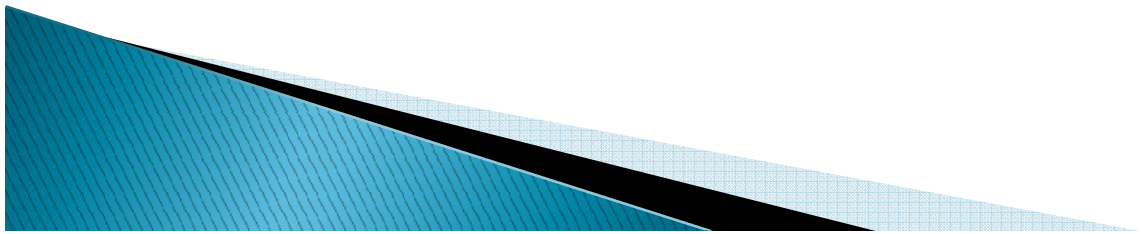
The Element Interface

public interface Element
extends Node

- Some methods in the Element interface

public NodeList getElementsByTagName(String name)

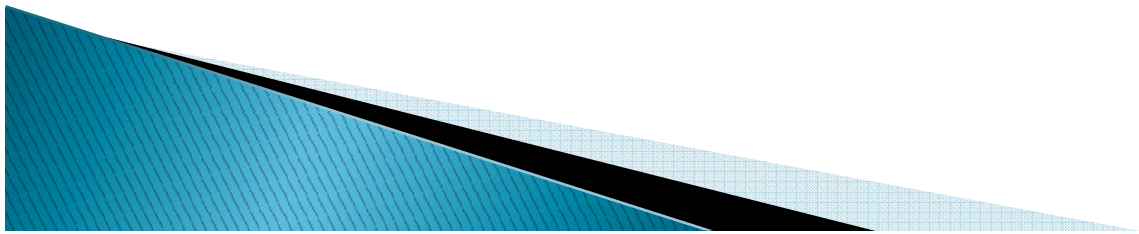
Returns a NodeList of all descendant elements with a given tag name, in the order in which they would be encountered in a preorder traversal of the Element tree..



The CharacterData Interface

public interface CharacterData
extends Node

The CharacterData interface extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to CharacterData, though Text and others do inherit the interface from it. All offsets in this interface start from 0.



The CharacterData Interface

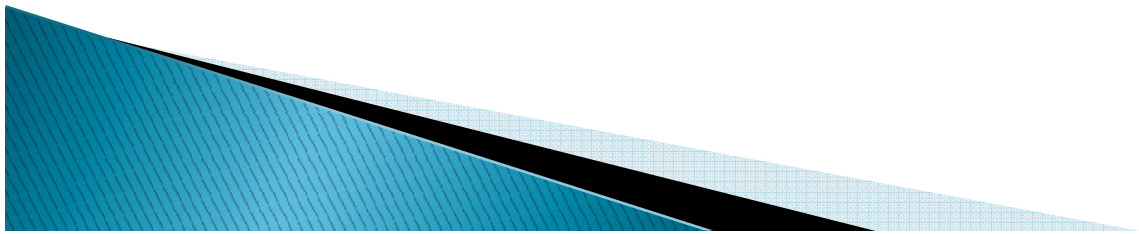
public interface CharacterData
extends Node

An example method:

public String getData()

Returns the value of the the character data of the node that implements this interface. The Text interface extends CharacterData.

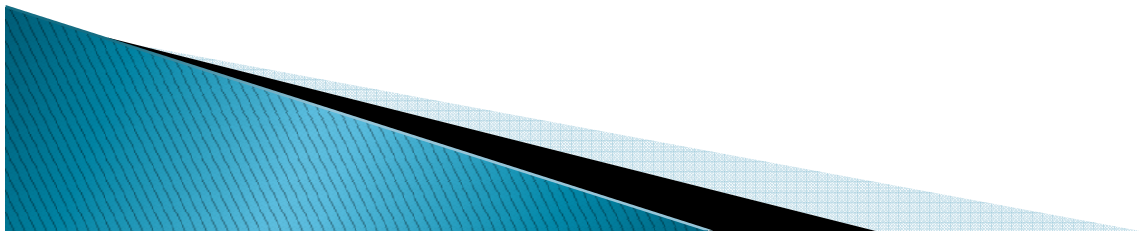
public void setData(String data) is also available.



The Document Interface

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.



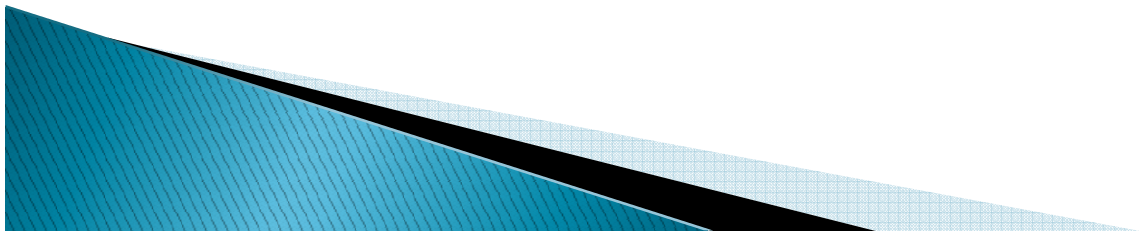
The Document Interface

public interface Document
extends Node

Some methods:

public Element getDocumentElement()

Returns the value of the documentElement property. This is a convenience attribute that allows direct access to the child node that is the root element of the document. For HTML documents, this is the element with the tagName "HTML".

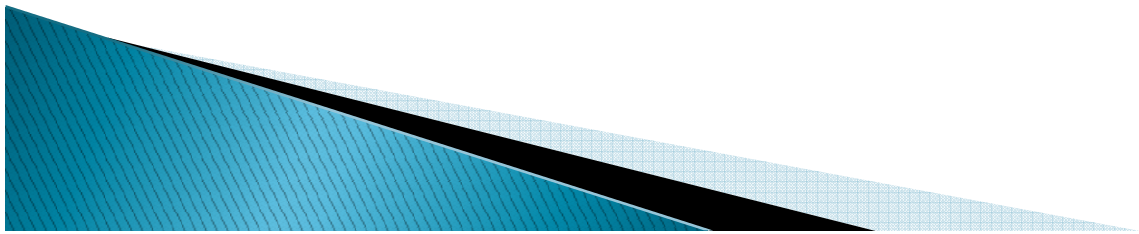


The Document Interface

Some methods:

public NodeList getElementsByTagName(String tagname)

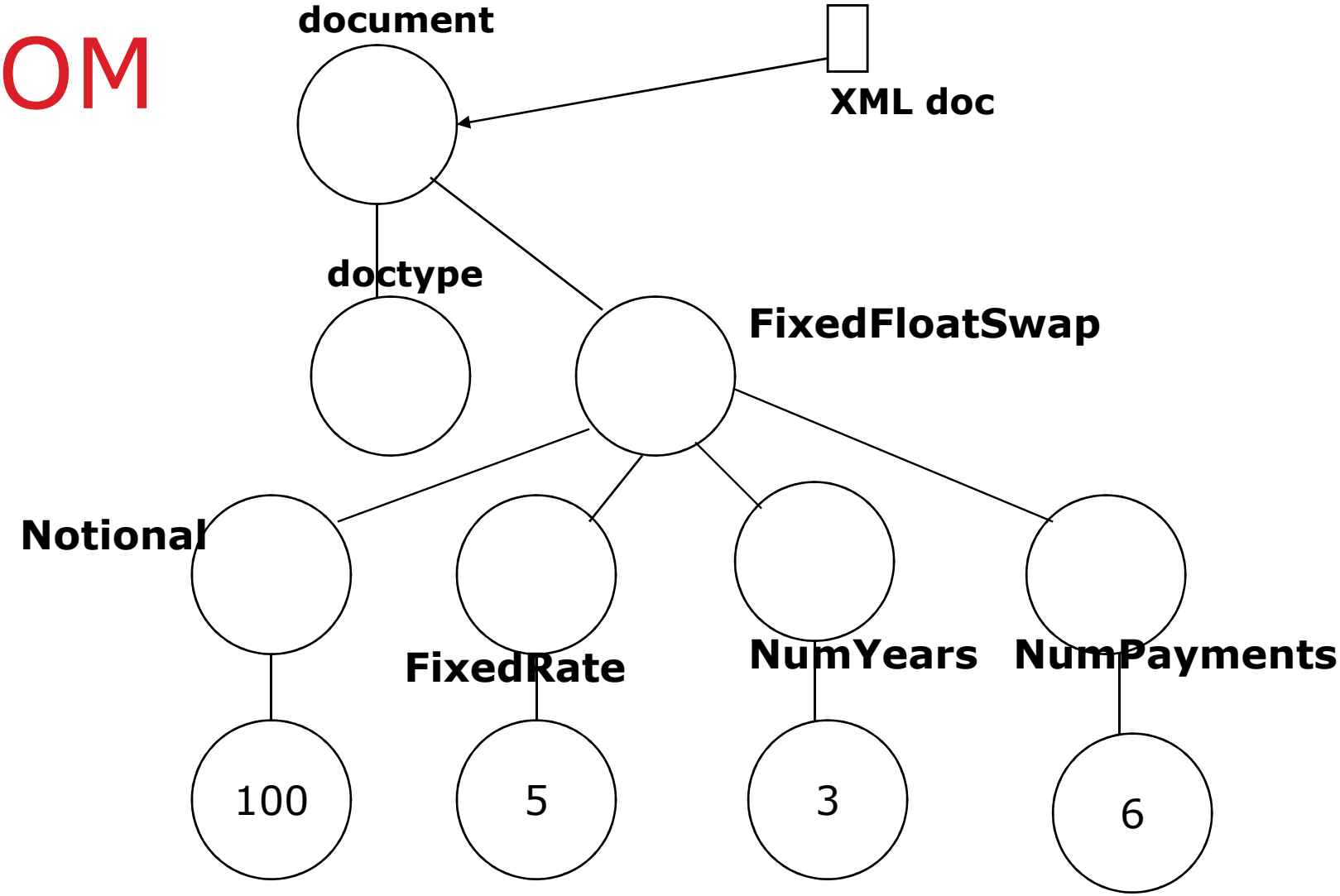
Returns a NodeList of all the Elements with a given tag name in the order in which the would be encountered in a preorder traversal of the Document tree.
Parameters:
tagname - The name of the tag to match on. The special value "*" matches all tags.
Returns:
A new NodeList object containing all the matched Elements.



FixedFloatSwap.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE FixedFloatSwap SYSTEM "FixedFloatSwap.dtd">
<FixedFloatSwap>
    <Notional>100</Notional>
    <Fixed_Rate>5</Fixed_Rate>
    <NumYears>3</NumYears>
    <NumPayments>6</NumPayments>
</FixedFloatSwap>
```

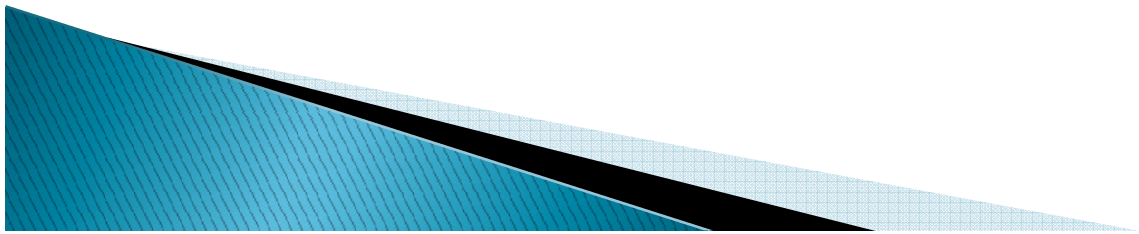
XML DOM



FixedFloatSwap.xml

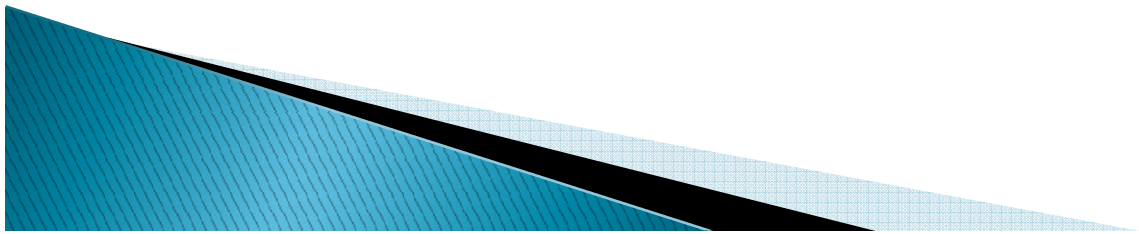
An Example

```
import java.io.File;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```



An Example

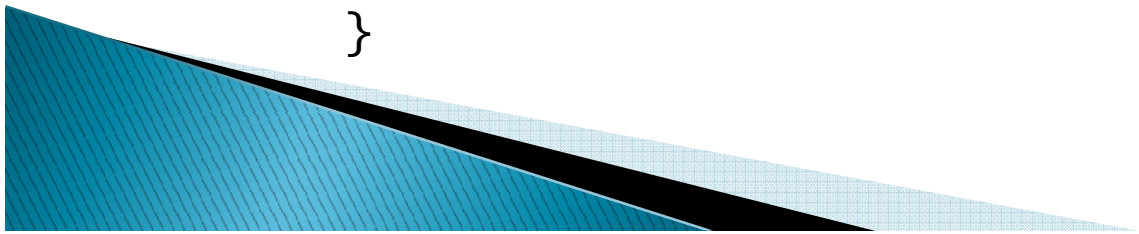
```
public class Simulator3
{
   public static void main(String argv[]) {
          Document doc;
          if(argv.length != 1 ) {
                 System.err.println("usage: java Simulator3 documentname");
                 System.exit(1);
          }
          try {
              DocumentBuilderFactory docBuilderFactory =
                                    DocumentBuilderFactory.newInstance();

              DocumentBuilder docBuilder =
                                    docBuilderFactory.newDocumentBuilder();
```



An Example

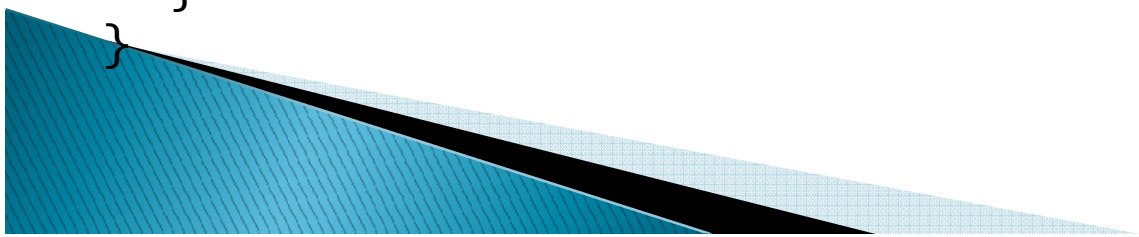
doc = docBuilder.parse(new File(argv[0]));

```
Element top = doc.getDocumentElement();
top.normalize();      // concatenate adjacent text nodes

NodeList elementList = top.getElementsByTagName("*");
int listLength = elementList.getLength();

for(int i = 0; i < listLength; i++) {

   Element e = (Element)elementList.item(i);
   System.out.print(e.getNodeName());
   Text t = (Text)e.getFirstChild();
   System.out.println(t.getNodeValue());

}
```



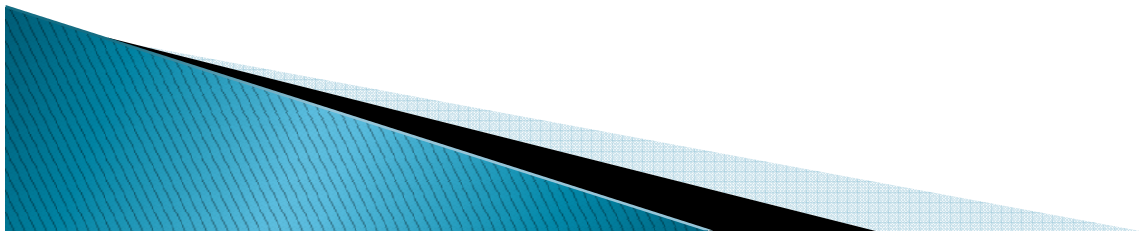
An Example

```
        }
        catch(SAXParseException err) {
              System.out.println("Parsing error" +
                                 ", line " + err.getLineNumber() +
                                  ", URI " + err.getSystemId());
              System.out.println("  " + err.getMessage());
         }
         catch(SAXException e) {
              Exception x = e.getException();
              ((x == null) ? e : x).printStackTrace();
         }
         catch (Throwable t) {
              t.printStackTrace();
         }
         System.exit(0);
    }
}
```



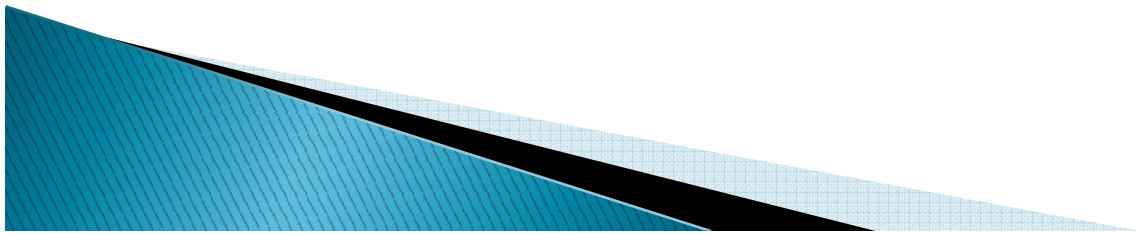
FixedFloatSwap.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE FixedFloatSwap SYSTEM "FixedFloatSwap.dtd">
<FixedFloatSwap>
    <Notional>100</Notional>
    <Fixed_Rate>5</Fixed_Rate>
    <NumYears>3</NumYears>
    <NumPayments>6</NumPayments>
</FixedFloatSwap>
```



Output

Notional100
Fixed_Rate5
NumYears3
NumPayments6

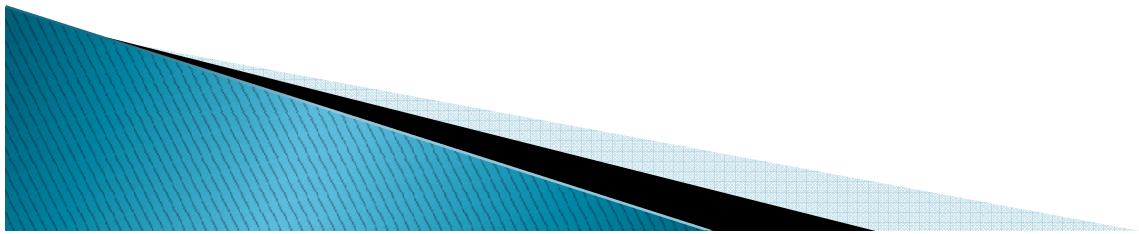


Building a DOM Tree From Scratch

MyGradeBook.xml

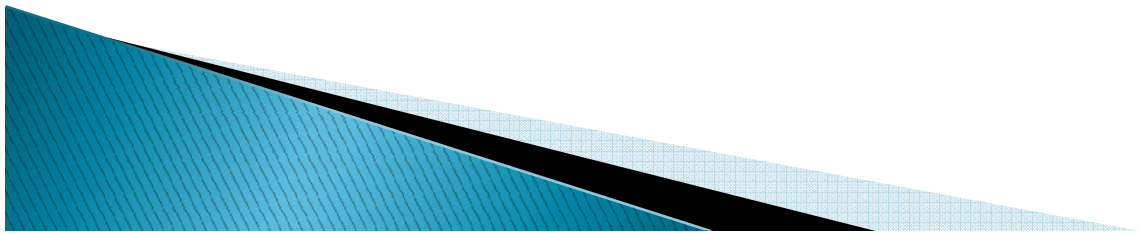
```
<?xml version="1.0" encoding="UTF-8"?>

<GradeBook>
  <Student>
    <Score>100</Score>
  </Student>
</GradeBook>
```

Let's create this file from within a java program.



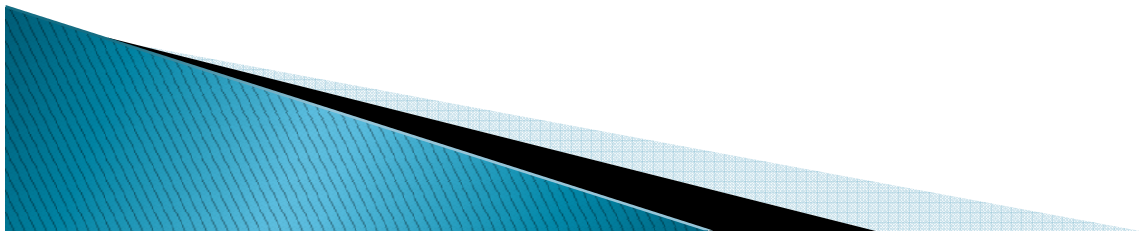
GOAL

```
C:\McCarthy\www\95-733\examples\dom>java DomExample

C:\McCarthy\www\95-733\examples\dom>type MyGradeBook.xml
<?xml version="1.0"?>
<GradeBook><Student><Score>100</Score></Student></GradeBook>
```



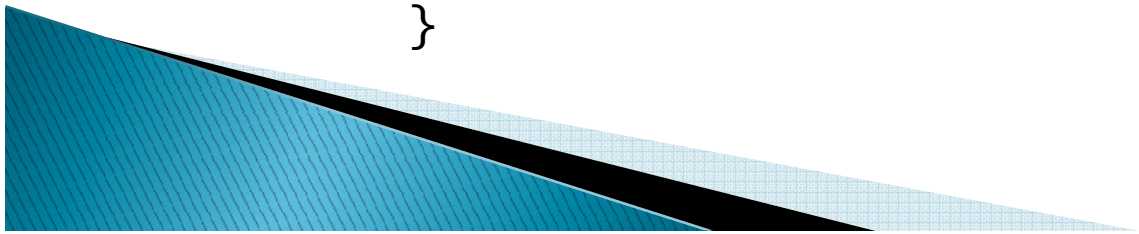
Building a DOM Tree From Scratch

```
// DomExample.java
// Building an xml document from scratch

import java.io.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.*;
import org.apache.xml.serialize.XMLSerializer;  // not standard
import org.apache.xml.serialize.OutputFormat;   // not standard
```



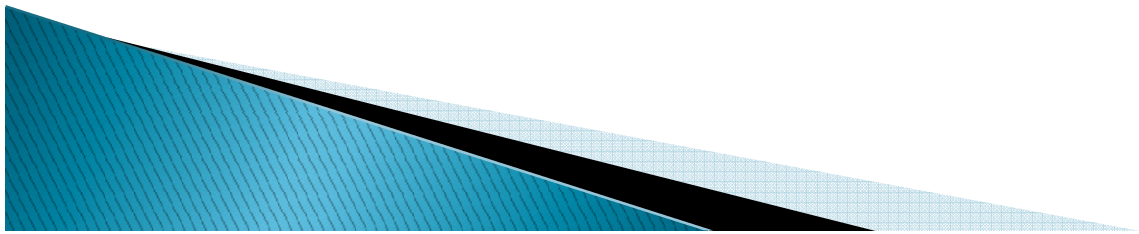
Building a DOM Tree From Scratch

```
public class DomExample {

        private Document document;
        public DomExample () {

            DocumentBuilderFactory factory =
                                DocumentBuilderFactory.newInstance();
            try {

                DocumentBuilder builder =
                                 factory.newDocumentBuilder();

                document = builder.newDocument();
            }
            catch (Throwable t) {
                    t.printStackTrace ();
             }
```



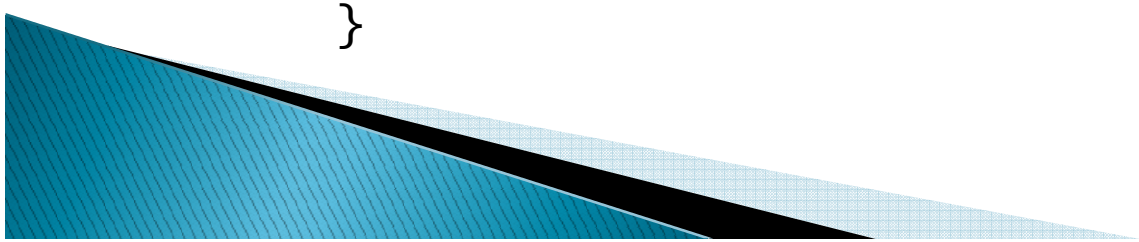
Building a DOM Tree From Scratch

```
// Ask the Document object for various types
// of nodes and add them to the tree.

Element root = document.createElement("GradeBook");
document.appendChild(root);

Element student = document.createElement("Student");
root.appendChild(student);

Element score = document.createElement("Score");
student.appendChild(score);

Text value = document.createTextNode("100");
score.appendChild(value);
```



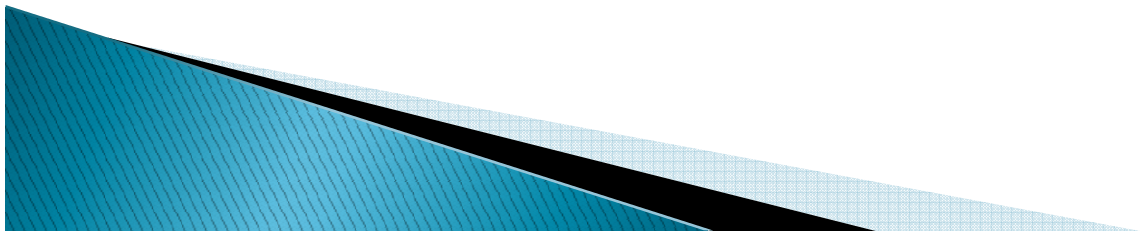
Building a DOM Tree From Scratch

```
        // Write the Document to disk using Xerces.
        try {

            FileOutputStream fos = new FileOutputStream(
                                                "MyGradeBook.xml");

            XMLSerializer xmlWriter =
                                                 new XMLSerializer(fos, null);

            xmlWriter.serialize(document);

        }
        catch(IOException ioe) {
            ioe.printStackTrace();
        }
    }
```



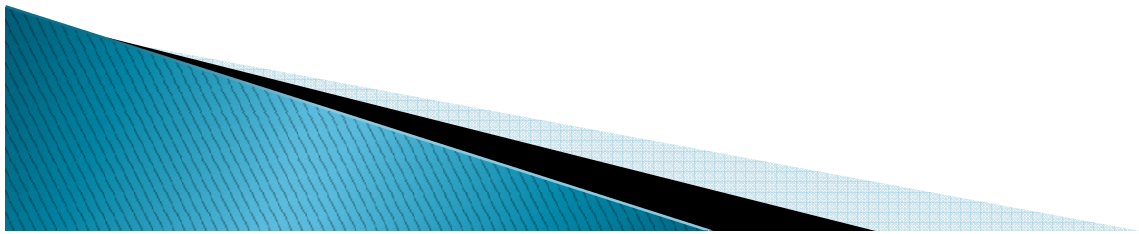
Building a DOM Tree From Scratch

```
public static void main(String a[]) {

        DomExample tree = new DomExample();
    }
}
```



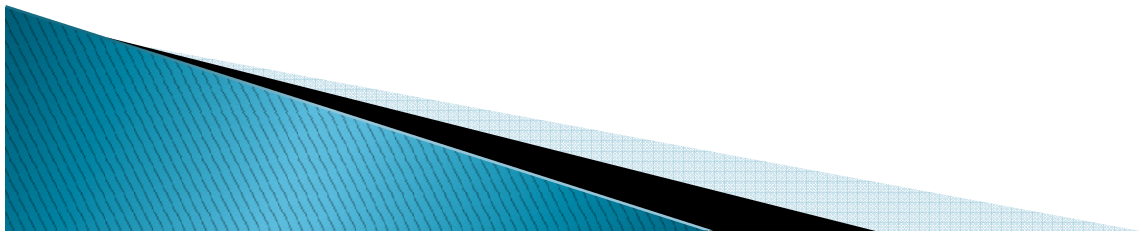
SAX Parser

- SAX (the Simple API for XML) is an event-based parser for XML documents.
- SAX 1.0 was released on May 11, 1998.
- The parser tells the application what is in the document by notifying the application of a stream of parsing events.
- Application then processes those events to act on data.



SAX Parser

- 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, it goes in one direction



Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC 94</recDate>
 <items>
   <item>
     <pnum>10506</pnum>
     <quantity>1</quantity>
   </item>
   <item>
     <pnum>10507</pnum>
     <quantity>1</quantity>
   </item>
   <item>
     <pnum>10508</pnum>
     <quantity>2</quantity>
   </item>
   <item>
     <pnum>10509</pnum>
     <quantity>3</quantity>
   </item>
 </items>
</order>
...
</orders>
```

Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

Parsing Event

```
<orders>
<order>
 <onum>1020</onum>
 <takenBy>1000</takenBy>
 <customer>1111</customer>
 <recDate>10-DEC 94</recDate>
 <items>
   <item>
     <pnum>10506</pnum>
     <quantity>1</quantity>
   </item>
    <item>
     <pnum>10507</pnum>
     <quantity>1</quantity>
    </item>
    <item>
     <pnum>10508</pnum>
     <quantity>2</quantity>
    </item>
    <item>
     <pnum>10509</pnum>
     <quantity>3</quantity>
    </item>
  </items>
</order>
...
</orders>
```

startDocument

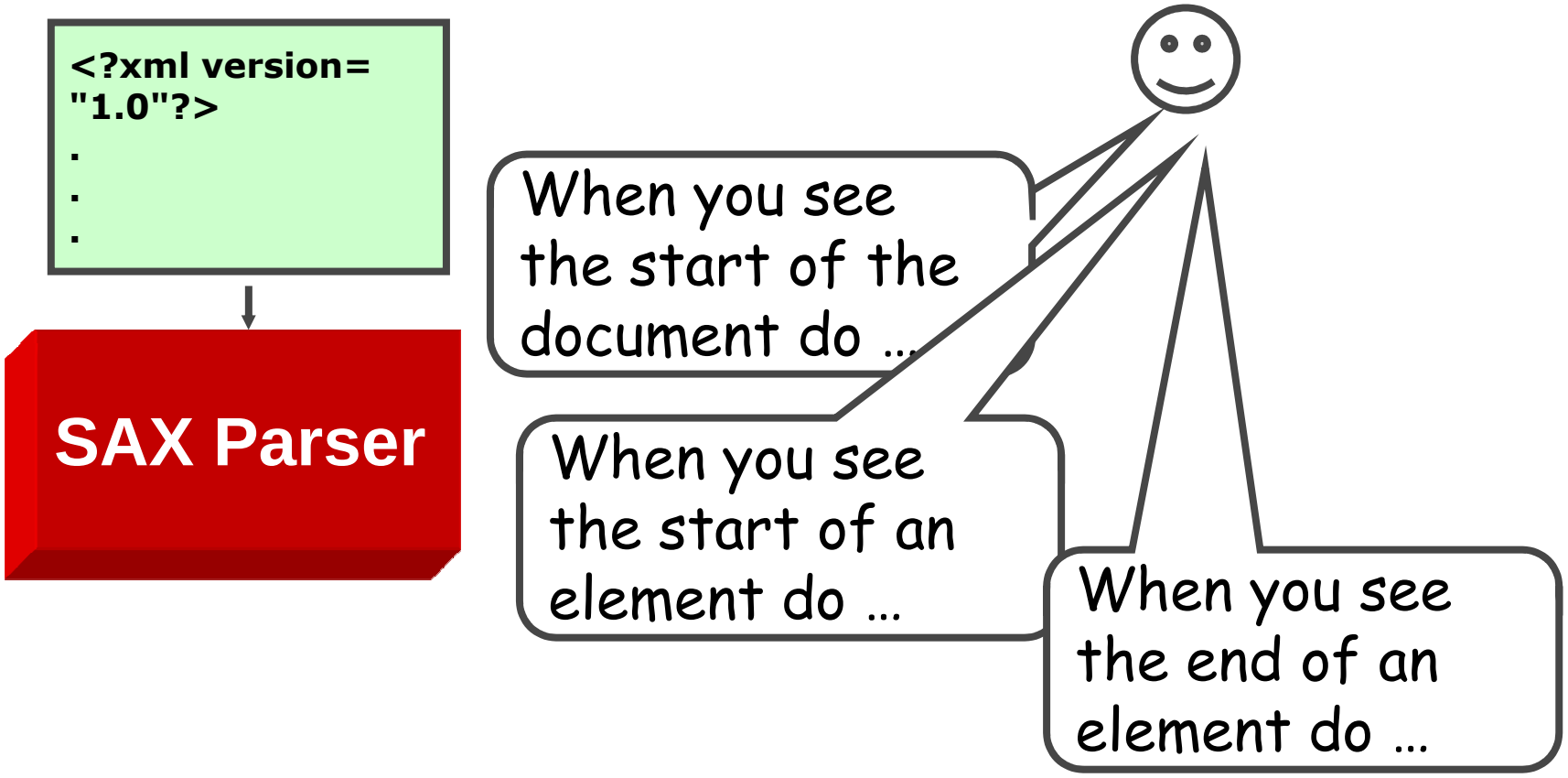
startElement

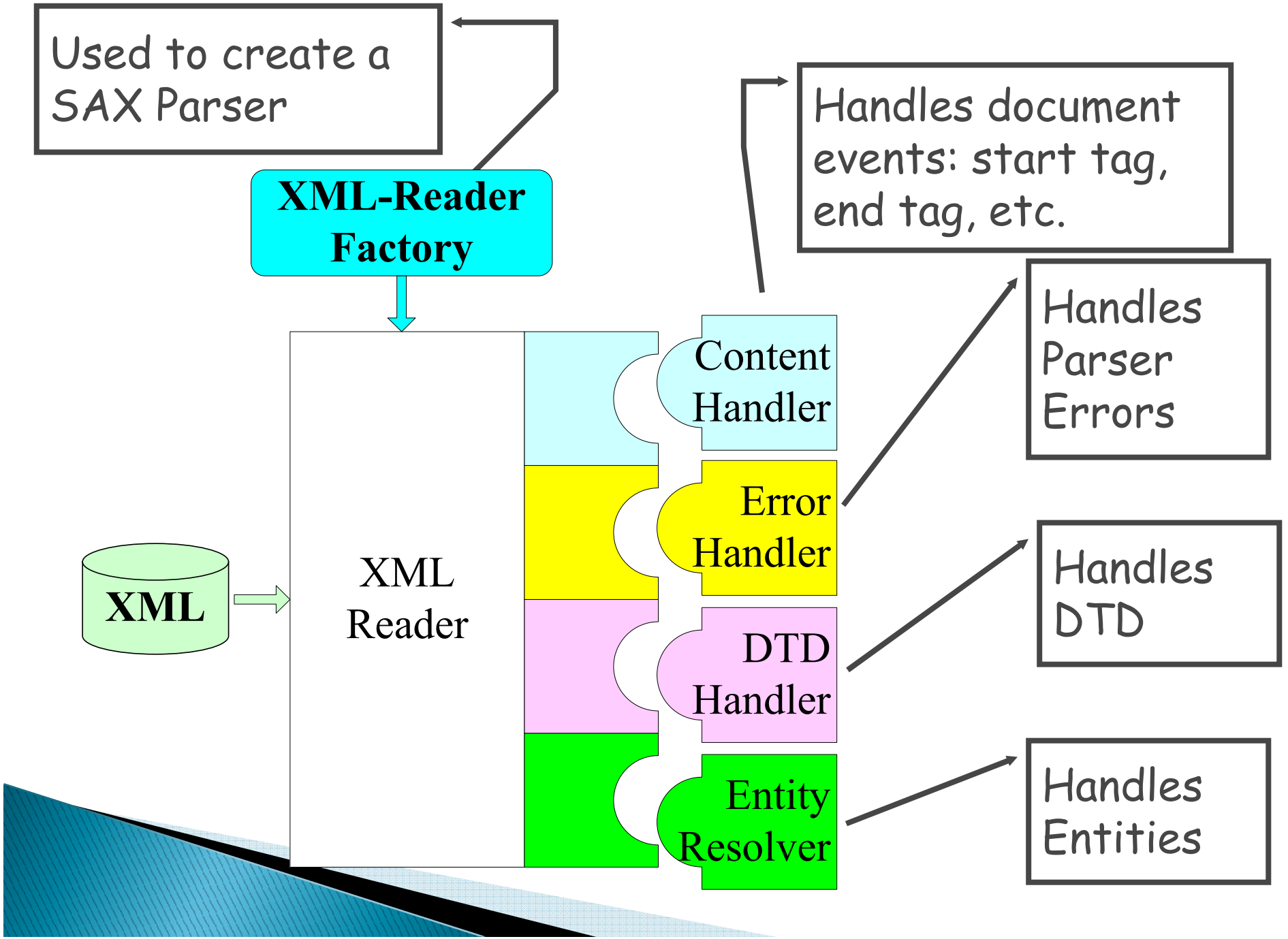
endElement

characters

endDocument

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
XML-Reader Factory
Handles document events: start tag, end tag, etc.
Handles Parser Errors
Handles DTD
Handles Entities
XML
XML Reader
Content Handler
Error Handler
DTD Handler
Entity Resolver

SAX API

Two important classes in the SAX API: `SAXParser` and `HandlerBase`.

A new `SAXParser` object is created by:

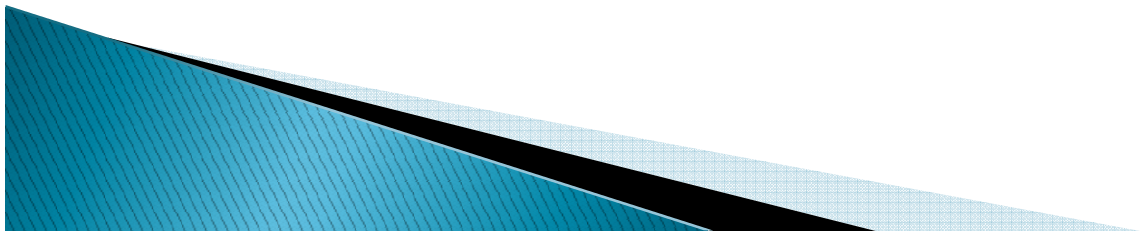
```
public SAXParser()
```

Register a SAX handler with the parser object to receive notifications of various parser events by:

```
public void setDocumentHandler(DocumentHandler h)
```

A similar method is available to register an error handler:

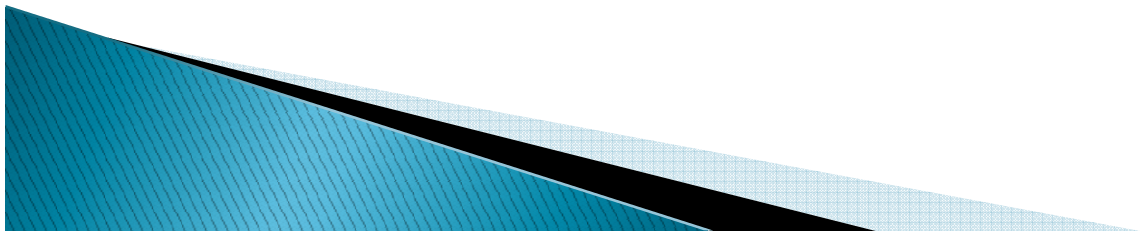
```
public void setErrorHandler(ErrorHandler h)
```



SAX API – Continued

- The `HandlerBase` class is a default base class for all handlers.
- It implements the default behavior for the various handlers.
- Application writers extend this class by rewriting the following event handler methods:

```
public void startDocument() throws SAXException
public void endDocument() throws SAXException
public void startElement() throws SAXException
public void endElement() throws SAXException
public void characters() throws SAXException
public void warning() throws SAXException
public void error() throws SAXException
```



Creating a SAX Parser

```
import org.xml.sax.*;
import oracle.xml.parser.v2.SAXParser;
public class SampleApp extends HandlerBase {
  // Global variables declared here
  static public void main(String [] args){
    Parser parser = new SAXParser();
    SampleApp app = new SampleApp();
    parser.setDocumentHandler(app);
    parser.setErrorHandler(app);
    try {
      parser.parse(createURL(args[0]).toString());
    } catch (SAXParseException e) {
        System.out.println(e.getMessage());
      }
  }
}
```

SAX API – Sample Application

Write a SAX Parser that reads the `orders.xml` file and extracts the various data items and creates SQL insert statements to insert the data into a relational database table.

```
//Global Variables
Vector itemNum = new Vector();
int numberOfRows, numberOfItems;
String elementEncountered, orderNumber, takenBy,
       customer, receivedDate, partNumber, quantity;
//elementEncountered holds most recent element name

public void startDocument() throws SAXException {
  //Print SQL comment, initialize variable
  System.out.println("--Start of SQL insert Statements");
  itemNum.setSize(1);
  numberOfRows = 0;
}//end startDocument
```

SAX API – Sample Application continued

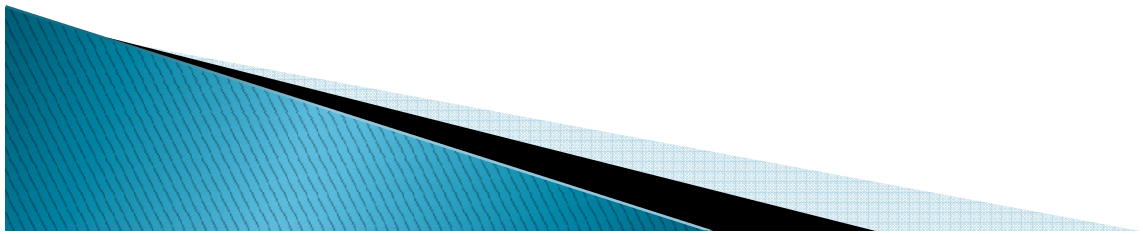
```
public void startElement(String name,
            AttributeList atts) throws SAXException {
  elementEncountered = name;
  if (name.equalsIgnoreCase("items")) {
    numberOfItems = 0;
  }//end if statement
}//end startElement

public void characters(char[] cbuf, int start, int len) {
  if (elementEncountered.equals("orderNumber")
    orderNumber = new String(cbuf,start,len);
  else if(elementEncountered.equals("takenBy")) {
    takenBy = new String(cbuf,start,len);
  ...
}//end characters
```

SAX API – Sample Application continued

```
public void endElement(String name) throws SAXException{
  if (name.equalsIgnoreCase("item")) {
    numberOfItems++;
    if (numberOfItems == 1) { // first item; create orders row
      System.out.println(
        "insert into orders values('"+
          orderNumber +"','"+ customer +"','"+
          takenBy +"','"+ receivedDate +"','null');");
      }
      System.out.println("insert into odetails values('"+
          orderNumber +"','"+ partNumber +"','"+
          quantity +"');");
  }//end if statement
  if (name.equalsIgnoreCase("items")) {
    System.out.println("--------");
  }
}//end endElement
```

SAX API – Sample Application continued

```
public void endDocument(){
  System.out.println("End of SQL insert statements.");
}//end endDocument

public void warning(SAXParseException e)
            throws SAXException {
  System.out.println("Warning:"+ e.getMessage());
}

public void error(SAXParseException e)
            throws SAXException{
  throw new SAXException(e.getMessage());
}
```

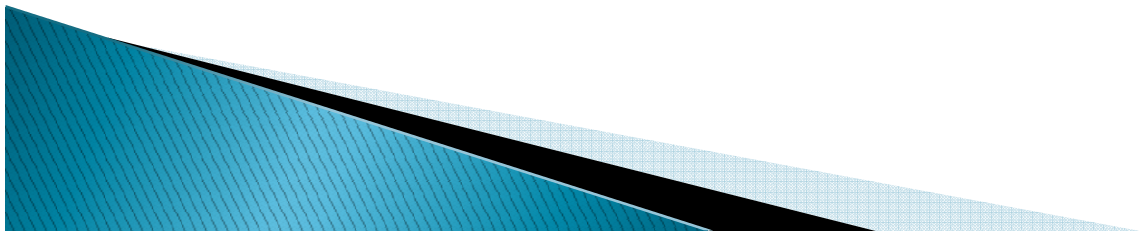


Difference between SAX and DOM

- DOM reads the entire XML document into memory and stores it as a tree data structure
- SAX reads the XML document and sends an event for each element that it encounters
- Consequences:
  - DOM provides “random access” into the XML document
  - SAX provides *only* sequential access to the XML document
  - DOM is slow and requires huge amounts of memory, so it cannot be used for large XML documents
  - SAX is fast and requires very little memory, so it can be used for huge documents (or large numbers of documents)
    - This makes SAX much more popular for web sites
  - Some DOM implementations have methods for *changing* the XML document in memory; SAX implementations do not

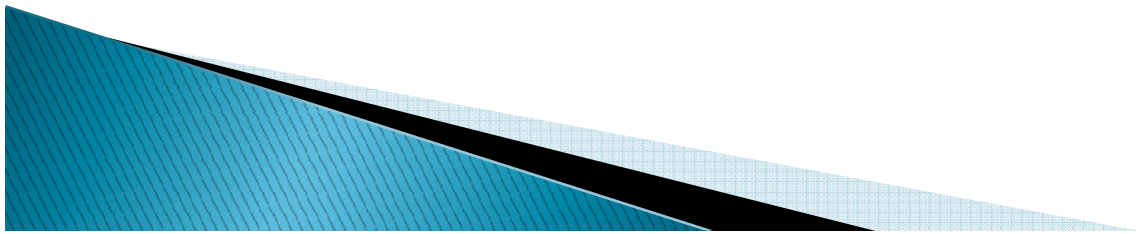
Programming using SAX is Difficult

- 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*?



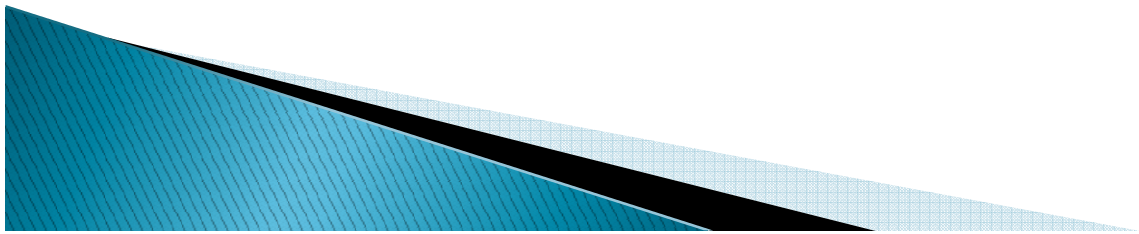
Node Navigation

- 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



Thank You