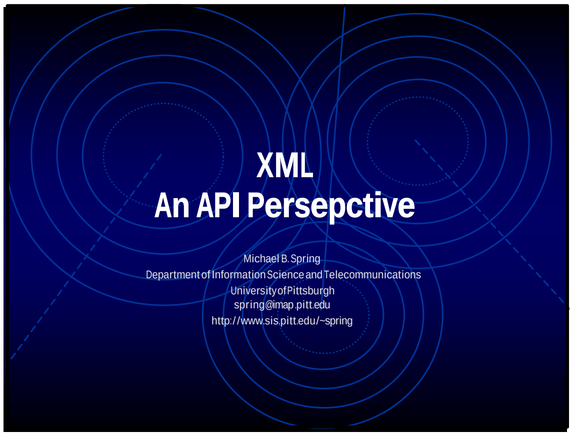
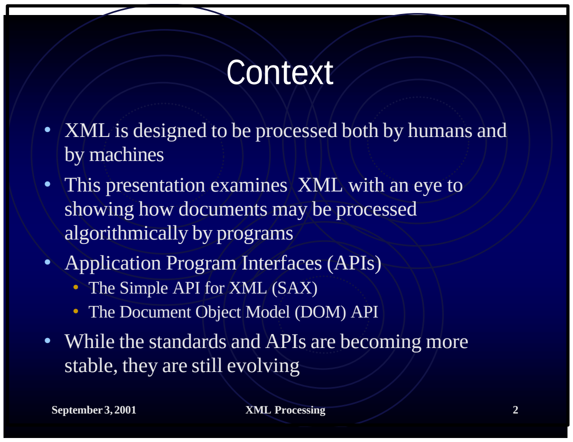


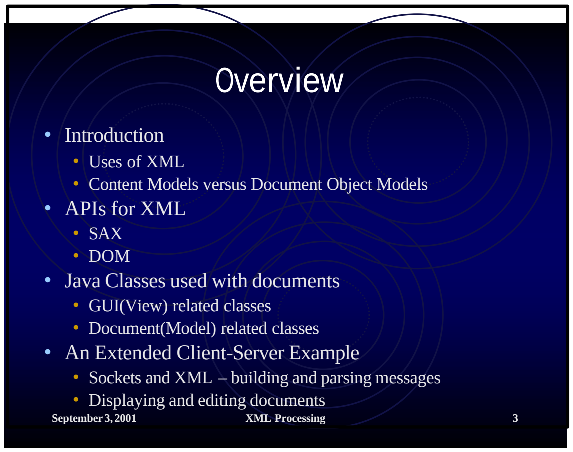
XML
An API Persepctive

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring



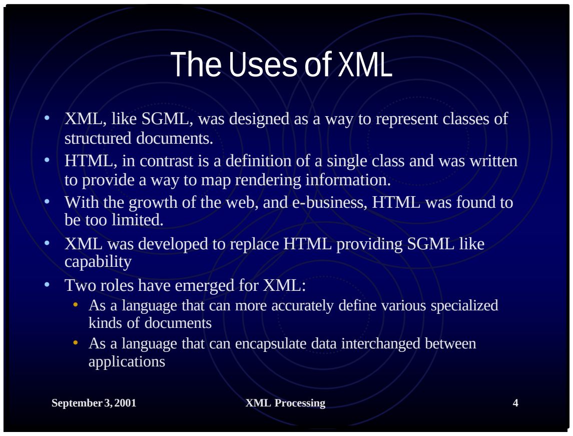
Context

- XML is designed to be processed both by humans and by machines
- This presentation examines XML with an eye to showing how documents may be processed algorithmically by programs
- Application Program Interfaces (APIs)
  - The Simple API for XML (SAX)
  - The Document Object Model (DOM) API
- While the standards and APIs are becoming more stable, they are still evolving



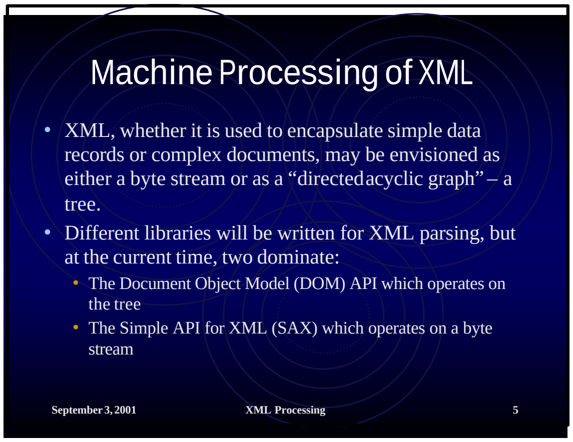
Overview

- Introduction
  - Uses of XML
  - Content Models versus Document Object Models
- APIs for XML
  - SAX
  - DOM
- Java Classes used with documents
  - GUI(View) related classes
  - Document(Model) related classes
- An Extended Client-Server Example
  - Sockets and XML – building and parsing messages
  - Displaying and editing documents



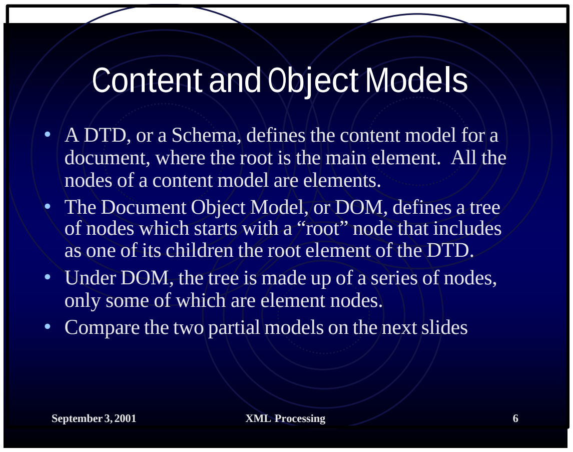
The Uses of XML

- XML, like SGML, was designed as a way to represent classes of structured documents.
- HTML, in contrast is a definition of a single class and was written to provide a way to map rendering information.
- With the growth of the web, and e-business, HTML was found to be too limited.
- XML was developed to replace HTML providing SGML like capability
- Two roles have emerged for XML:
  - As a language that can more accurately define various specialized kinds of documents
  - As a language that can encapsulate data interchanged between applications



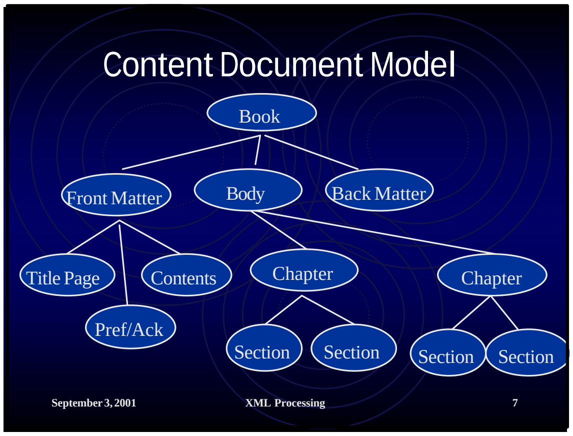
Machine Processing of XML

- XML, whether it is used to encapsulate simple data records or complex documents, may be envisioned as either a byte stream or as a “directedacyclic graph” – a tree.
- Different libraries will be written for XML parsing, but at the current time, two dominate:
  - The Document Object Model (DOM) API which operates on the tree
  - The Simple API for XML (SAX) which operates on a byte stream



Content and Object Models

- A DTD, or a Schema, defines the content model for a document, where the root is the main element. All the nodes of a content model are elements.
- The Document Object Model, or DOM, defines a tree of nodes which starts with a “root” node that includes as one of its children the root element of the DTD.
- Under DOM, the tree is made up of a series of nodes, only some of which are element nodes.
- Compare the two partial models on the next slides



Content Document Model

Book

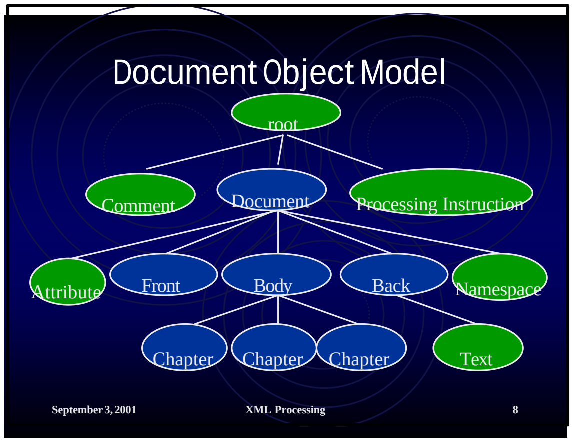
Front Matter Body Back Matter

Title Page Contents Chapter Chapter

Pref/Ack

Section Section Section Section

September 3, 2001 XML Processing 7



Document Object Model

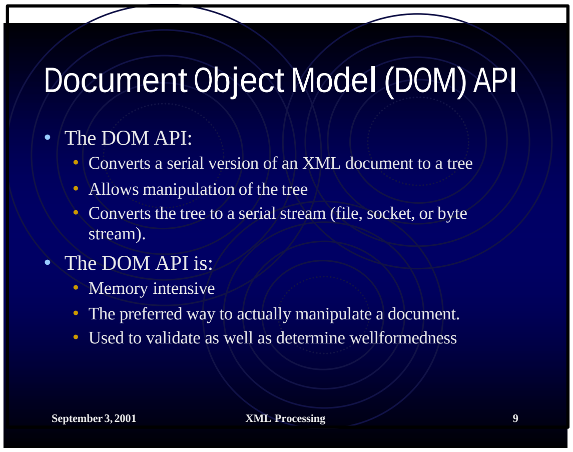
root

Comment Document Processing Instruction

Attribute Front Body Back Namespace

Chapter Chapter Chapter Text

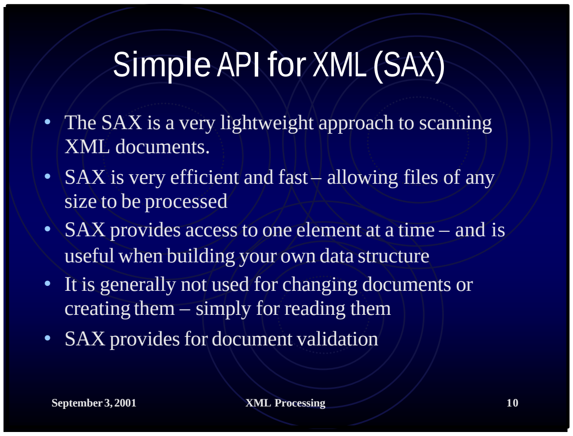
September 3, 2001 XML Processing 8



Document Object Model (DOM) API

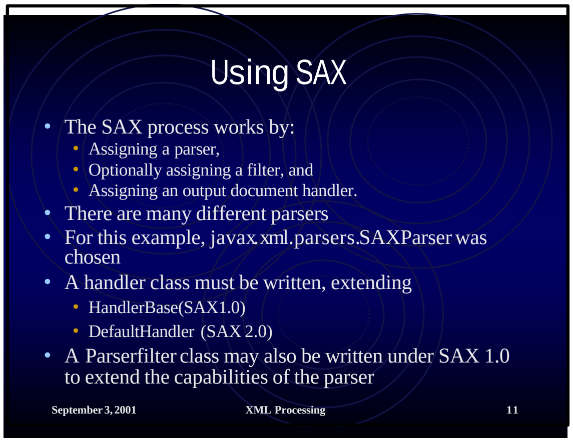
- The DOM API:
  - Converts a serial version of an XML document to a tree
  - Allows manipulation of the tree
  - Converts the tree to a serial stream (file, socket, or byte stream).
- The DOM API is:
  - Memory intensive
  - The preferred way to actually manipulate a document.
  - Used to validate as well as determine wellformedness

September 3, 2001 XML Processing 9



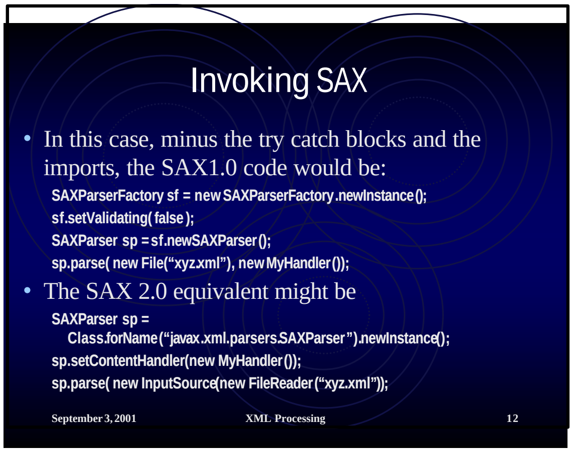
Simple API for XML (SAX)

- The SAX is a very lightweight approach to scanning XML documents.
- SAX is very efficient and fast – allowing files of any size to be processed
- SAX provides access to one element at a time – and is useful when building your own data structure
- It is generally not used for changing documents or creating them – simply for reading them
- SAX provides for document validation



Using SAX

- The SAX process works by:
  - Assigning a parser,
  - Optionally assigning a filter, and
  - Assigning an output document handler.
- There are many different parsers
- For this example, javax.xml.parsers.SAXParser was chosen
- A handler class must be written, extending
  - HandlerBase(SAX1.0)
  - DefaultHandler (SAX 2.0)
- A Parserfilter class may also be written under SAX 1.0 to extend the capabilities of the parser



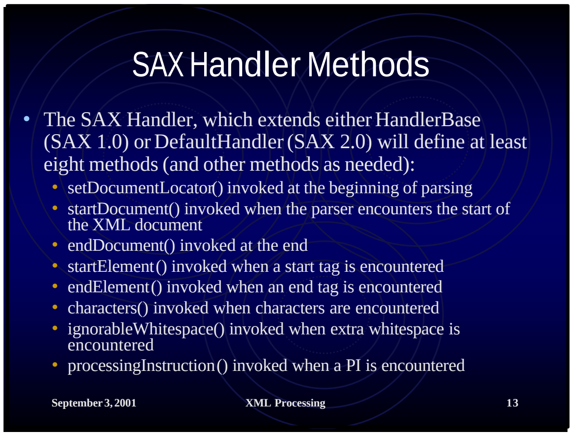
Invoking SAX

- In this case, minus the try catch blocks and the imports, the SAX1.0 code would be:

```
SAXParserFactory sf = new SAXParserFactory.newInstance();
sf.setValidating( false );
SAXParser sp = sf.newSAXParser();
sp.parse( new File("xyz.xml"), new MyHandler());
```

- The SAX 2.0 equivalent might be

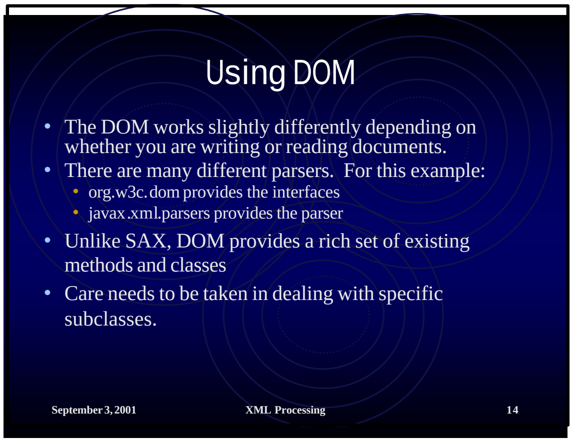
```
SAXParser sp =
   Class.forName("javax.xml.parsers.SAXParser").newInstance();
sp.setContentHandler(new MyHandler());
sp.parse( new InputSource(new FileReader("xyz.xml"));
```



SAX Handler Methods

- The SAX Handler, which extends either HandlerBase (SAX 1.0) or DefaultHandler (SAX 2.0) will define at least eight methods (and other methods as needed):
  - setDocumentLocator() invoked at the beginning of parsing
  - startDocument() invoked when the parser encounters the start of the XML document
  - endDocument() invoked at the end
  - startElement() invoked when a start tag is encountered
  - endElement() invoked when an end tag is encountered
  - characters() invoked when characters are encountered
  - ignorableWhitespace() invoked when extra whitespace is encountered
  - processingInstruction() invoked when a PI is encountered

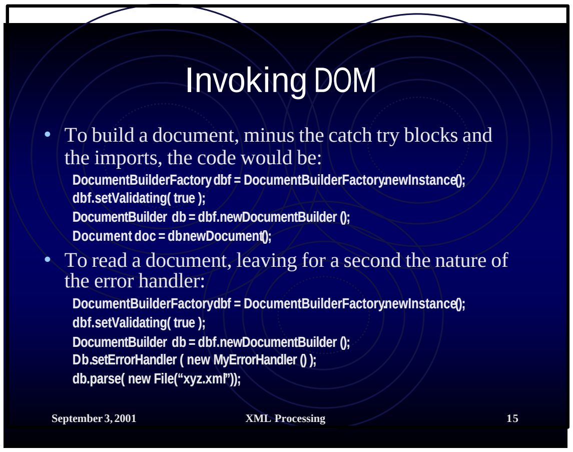
September 3, 2001 XML Processing 13



Using DOM

- The DOM works slightly differently depending on whether you are writing or reading documents.
- There are many different parsers. For this example:
  - org.w3c.dom provides the interfaces
  - javax.xml.parsers provides the parser
- Unlike SAX, DOM provides a rich set of existing methods and classes
- Care needs to be taken in dealing with specific subclasses.

September 3, 2001 XML Processing 14



Invoking DOM

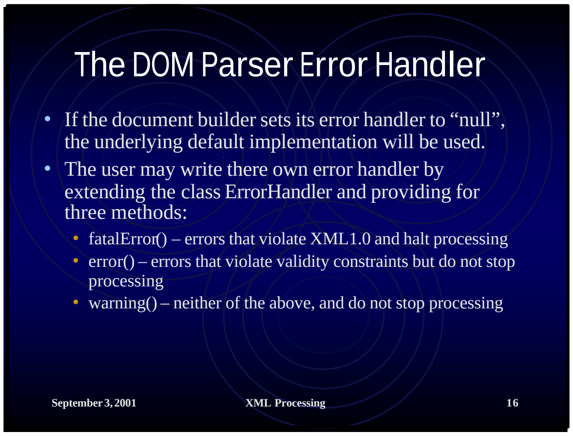
- To build a document, minus the catch try blocks and the imports, the code would be:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating( true );
DocumentBuilder  db = dbf.newDocumentBuilder ();
Document doc = dbnewDocument();
```

- To read a document, leaving for a second the nature of the error handler:

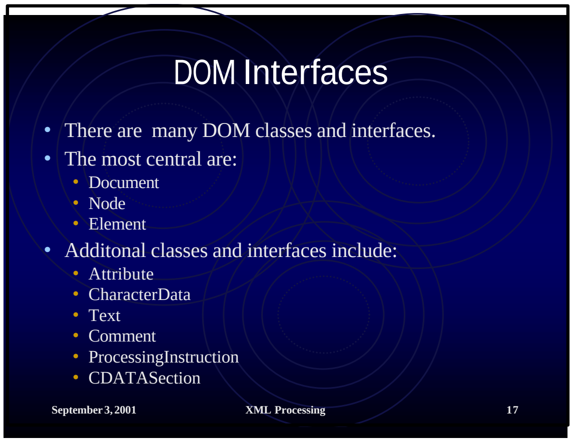
```
DocumentBuilderFactorydbf = DocumentBuilderFactory.newInstance();
dbf.setValidating( true );
DocumentBuilder  db = dbf.newDocumentBuilder ();
Db.setErrorHandler ( new MyErrorHandler () );
db.parse( new File(“xyz.xml”));
```

September 3, 2001 XML Processing 15



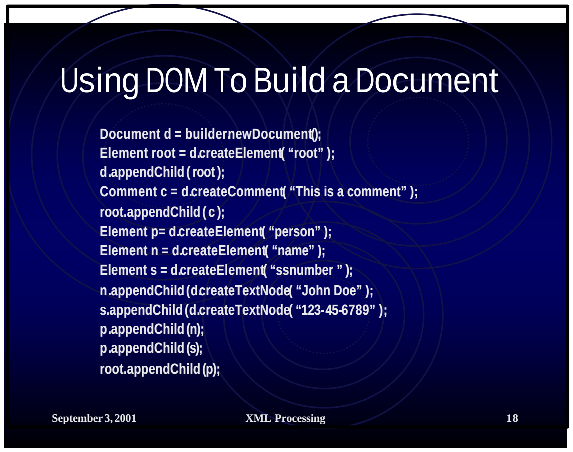
The DOM Parser Error Handler

- If the document builder sets its error handler to “null”, the underlying default implementation will be used.
- The user may write there own error handler by extending the class ErrorHandler and providing for three methods:
  - fatalError() – errors that violate XML1.0 and halt processing
  - error() – errors that violate validity constraints but do not stop processing
  - warning() – neither of the above, and do not stop processing



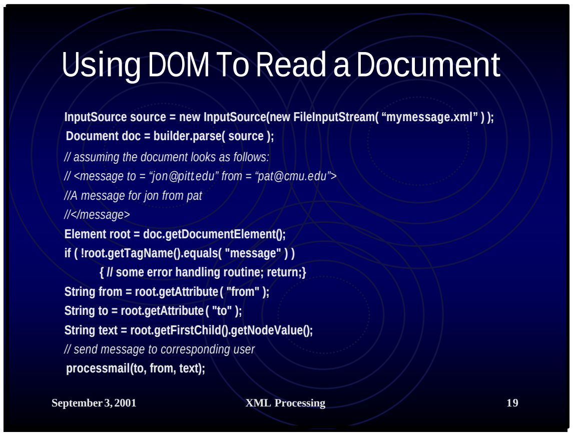
DOM Interfaces

- There are many DOM classes and interfaces.
- The most central are:
  - Document
  - Node
  - Element
- Additonal classes and interfaces include:
  - Attribute
  - CharacterData
  - Text
  - Comment
  - ProcessingInstruction
  - CDATASection



Using DOM To Build a Document

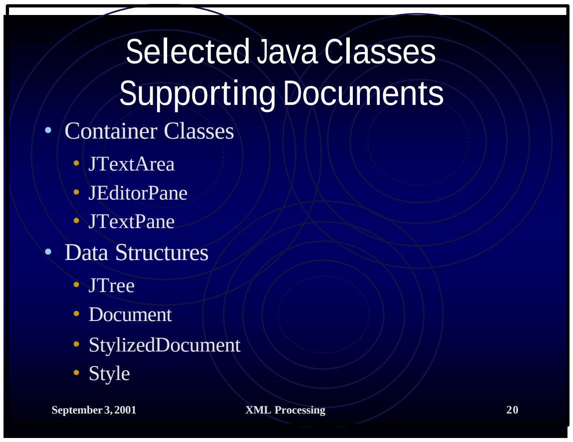
```
Document d = builder.newDocument();
Element root = d.createElement( “root” );
d.appendChild ( root );
Comment c = d.createComment( “This is a comment” );
root.appendChild ( c );
Element p= d.createElement( “person” );
Element n = d.createElement( “name” );
Element s = d.createElement( “ssnumber ” );
n.appendChild (d.createTextNode( “John Doe” );
s.appendChild (d.createTextNode( “123-45-6789” );
p.appendChild (n);
p.appendChild (s);
root.appendChild (p);
```



Using DOM To Read a Document

```
InputSource source = new InputSource(new FileInputStream( “mymessage.xml” ) );
 Document doc = builder.parse( source );
// assuming the document looks as follows:
// <message to = “jon@pitt.edu” from = “pat@cmu.edu”>
//A message for jon from pat
//</message>
Element root = doc.getDocumentElement();
if ( !root.getTagName().equals( "message" ) )
          { // some error handling routine; return;}
String from = root.getAttribute( "from" );
String to = root.getAttribute( "to" );
String text = root.getFirstChild().getNodeValue();
// send message to corresponding user
 processmail(to, from, text);
```

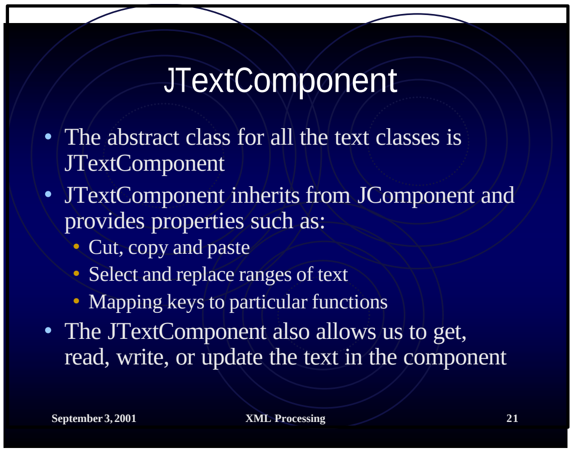
September 3, 2001 XML Processing 19



Selected Java Classes Supporting Documents

- Container Classes
  - JTextArea
  - JEditorPane
  - JTextPane
- Data Structures
  - JTree
  - Document
  - StylizedDocument
  - Style

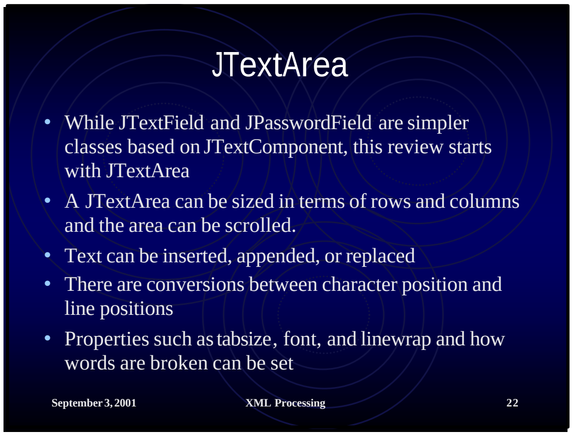
September 3, 2001 XML Processing 20



JTextComponent

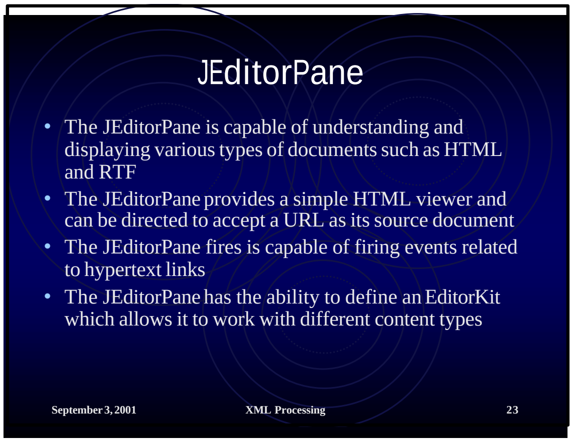
- The abstract class for all the text classes is JTextComponent
- JTextComponent inherits from JComponent and provides properties such as:
  - Cut, copy and paste
  - Select and replace ranges of text
  - Mapping keys to particular functions
- The JTextComponent also allows us to get, read, write, or update the text in the component

September 3, 2001 XML Processing 21



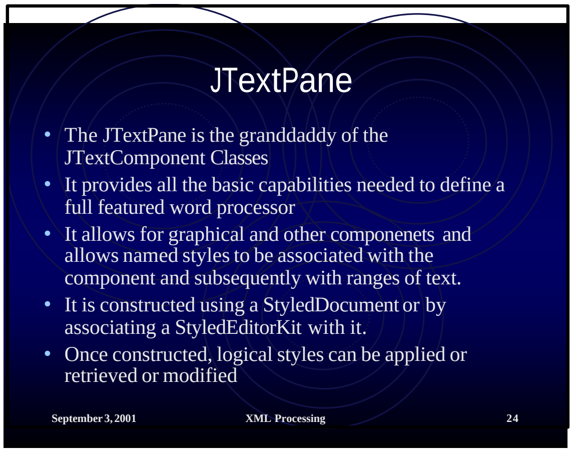
JTextArea

- While JTextField and JPasswordField are simpler classes based on JTextComponent, this review starts with JTextArea
- A JTextArea can be sized in terms of rows and columns and the area can be scrolled.
- Text can be inserted, appended, or replaced
- There are conversions between character position and line positions
- Properties such as tabsize, font, and linewrap and how words are broken can be set



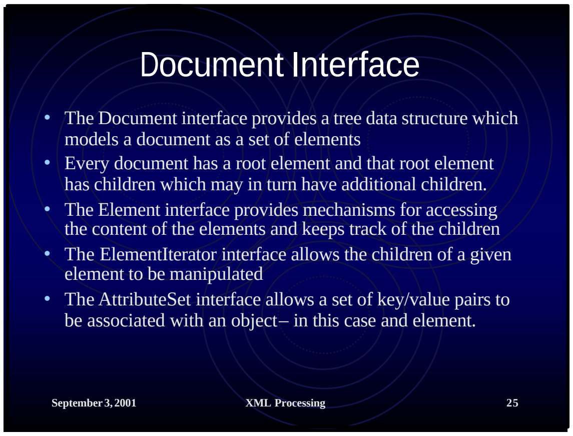
JEditorPane

- The JEditorPane is capable of understanding and displaying various types of documents such as HTML and RTF
- The JEditorPane provides a simple HTML viewer and can be directed to accept a URL as its source document
- The JEditorPane fires is capable of firing events related to hypertext links
- The JEditorPane has the ability to define an EditorKit which allows it to work with different content types



JTextPane

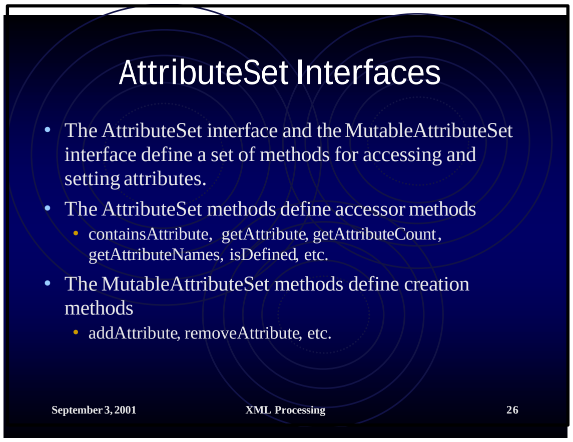
- The JTextPane is the granddaddy of the JTextComponent Classes
- It provides all the basic capabilities needed to define a full featured word processor
- It allows for graphical and other componenets and allows named styles to be associated with the component and subsequently with ranges of text.
- It is constructed using a StyledDocument or by associating a StyledEditorKit with it.
- Once constructed, logical styles can be applied or retrieved or modified



Document Interface

- The Document interface provides a tree data structure which models a document as a set of elements
- Every document has a root element and that root element has children which may in turn have additional children.
- The Element interface provides mechanisms for accessing the content of the elements and keeps track of the children
- The ElementIterator interface allows the children of a given element to be manipulated
- The AttributeSet interface allows a set of key/value pairs to be associated with an object– in this case and element.

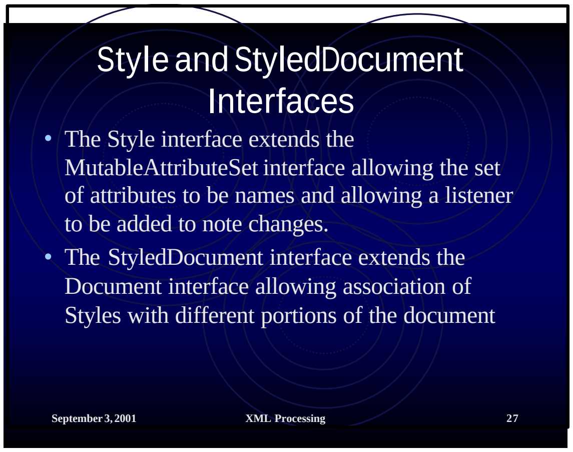
September 3, 2001 XML Processing 25



AttributeSet Interfaces

- The AttributeSet interface and the MutableAttributeSet interface define a set of methods for accessing and setting attributes.
- The AttributeSet methods define accessor methods
  - containsAttribute, getAttribute, getAttributeCount, getAttributeNames, isDefined, etc.
- The MutableAttributeSet methods define creation methods
  - addAttribute, removeAttribute, etc.

September 3, 2001 XML Processing 26



Style and StyledDocument Interfaces

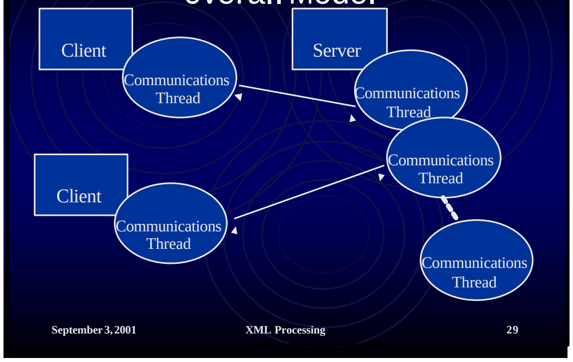
- The Style interface extends the MutableAttributeSet interface allowing the set of attributes to be names and allowing a listener to be added to note changes.
- The StyledDocument interface extends the Document interface allowing association of Styles with different portions of the document

September 3, 2001 XML Processing 27

Some Code Snipets

- The following slides provide a conceptual overview and a few pieces of code from a client server application for collaborative authoring.
- The code is written in Java, uses threads, and uses:
  - Dave Meggison's crimson classes
  - SUN's jaxp
  - W3C xerces parsers
  - SUN xlan parsers

Overall Model



Client

Communications Thread

Server

Communications Thread

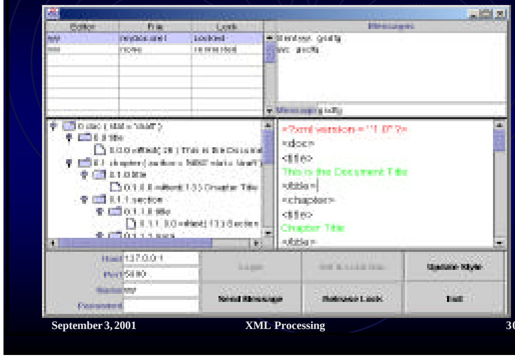
Communications Thread

Client

Communications Thread

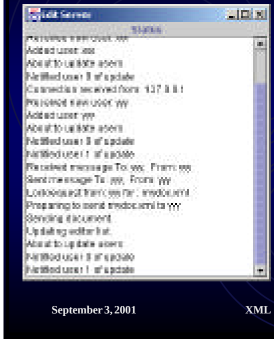
Communications Thread

The Client GUI



```
<?xml version="1.0"?>
<doc>
<title>
This is the Document Title
</title>
<chapter>
<title>
Chapter Title
</title>
```

The Server



```
Received from user: yyy
Added user: xxx
About to update users
Notified user 0 of update
Connection received from: 127.0.0.1
Received new user: yyy
Added user: yyy
About to update users
Notified user 0 of update
Notified user 1 of update
Received message To: yyy; From: xxx
Sent message To: yyy; From: yyy
Lookrequest from yyy for : mydoc.xml
Preparing to send mydoc.xml to yyy
Sending document
Updating editor list.
About to update users
Notified user 0 of update
Notified user 1 of update
```

- The server simply logs and keeps track of the activity of the clients
- A separate set of threads handles communications among the various clients

Message Construction

- For the application as a whole

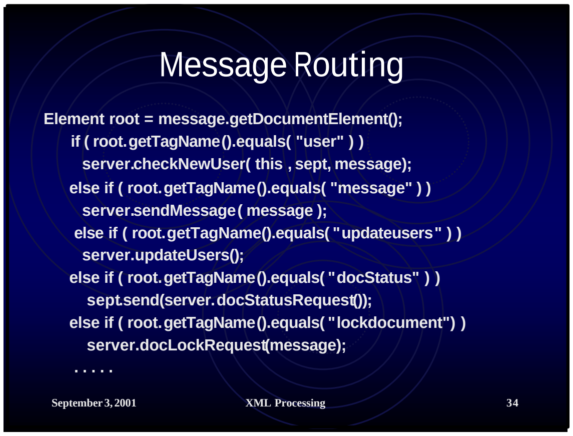
```
try   { // obtain the default parser
          factory = DocumentBuilderFactory.newInstance();
          // get DocumentBuilder
          builder = factory.newDocumentBuilder();   }
catch ( ParserConfigurationException pce ) {
          pce.printStackTrace();}
```

- To construct a simple document to be sent

```
Document login = builder.newDocument();
 Element root = login.createElement( "user" );
 login.appendChild( root );
 lp.set_tf_name(user);
 root.appendChild(login.createTextNode( user ) );
 send( login );
```

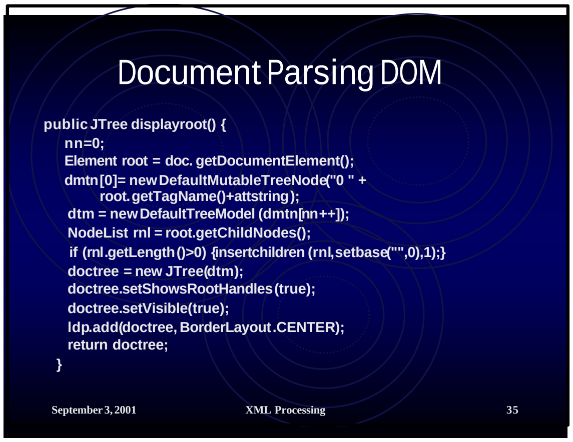
Sending a message

```
public void send( Document message )
 {byte end[]={0,0};    byte mt[]={1,1};
    try {
      // write to output stream
       output.write(mt); //1 indicates a text message
       TransformerFactory transformerFactory =
            TransformerFactory.newInstance();
       Transformer  serializer = transformerFactory.newTransformer ();
       serializer.transform( new DOMSource( message ),
            new StreamResult( output ) );
       output.write(end);
       output.flush(); }
    catch ( Exception e ) {  e.printStackTrace();   }
 }
```



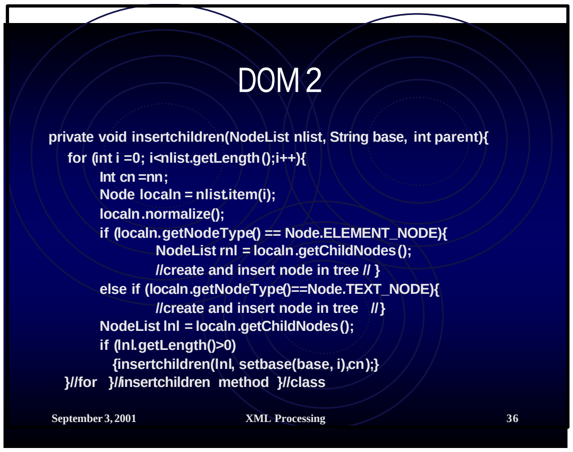
Message Routing

```
Element root = message.getDocumentElement();
    if ( root.getTagName().equals( "user" ) )
      server.checkNewUser( this , sept, message);
    else if ( root.getTagName().equals( "message" ) )
      server.sendMessage( message );
     else if ( root.getTagName().equals( "updateusers" ) )
      server.updateUsers();
    else if ( root.getTagName().equals( "docStatus" ) )
       sept.send(server.docStatusRequest());
    else if ( root.getTagName().equals( "lockdocument") )
       server.docLockRequest(message);
     . . . . .
```



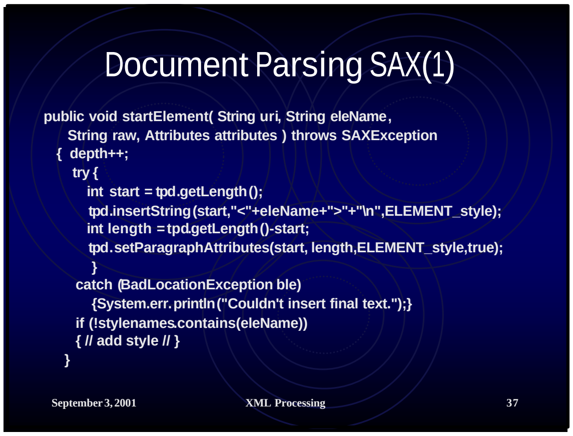
Document Parsing DOM

```
public JTree displayroot() {
   nn=0;
   Element root = doc. getDocumentElement();
   dmtn[0]= new DefaultMutableTreeNode("0 " +
        root. getTagName()+attstring);
    dtm = new DefaultTreeModel (dmtn[nn++]);
    NodeList rnl = root.getChildNodes();
     if (rnl.getLength()>0) {insertchildren (rnl,setbase("",0),1);}
    doctree = new JTree(dtm);
    doctree.setShowsRootHandles(true);
    doctree.setVisible(true);
    ldp.add(doctree, BorderLayout.CENTER);
    return doctree;
  }
```



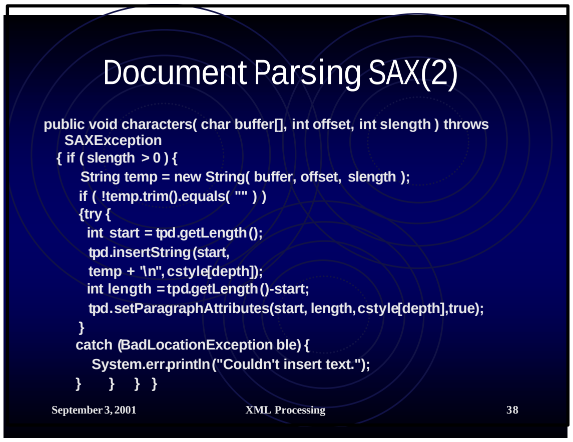
DOM 2

```
private void insertchildren(NodeList nlist, String base, int parent){
   for (int i =0; i<nlist.getLength();i++){
          Int cn =nn;
          Node localn = nlist.item(i);
          localn.normalize();
          if (localn.getNodeType() == Node.ELEMENT_NODE){
                   NodeList rnl = localn.getChildNodes();
                   //create and insert node in tree // }
          else if (localn.getNodeType()==Node.TEXT_NODE){
                   //create and insert node in tree   //}
          NodeList lnl = localn.getChildNodes();
          if (lnl.getLength()>0)
            {insertchildren(lnl, setbase(base, i),cn);}
  }//for  }//insertchildren  method  }//class
```



Document Parsing SAX(1)

```
public void startElement( String uri, String eleName,
    String raw, Attributes attributes ) throws SAXException
  {  depth++;
     try {
        int  start = tpd.getLength();
        tpd.insertString(start,"<"+eleName+">"+"\n",ELEMENT_style);
        int length = tpd.getLength()-start;
        tpd.setParagraphAttributes(start, length,ELEMENT_style,true);
         }
      catch (BadLocationException ble)
         {System.err.println("Couldn't insert final text.");}
      if (!stylenames.contains(eleName))
      { // add style // }
    }
```



Document Parsing SAX(2)

```
public void characters( char buffer[], int offset, int slength ) throws
    SAXException
  { if ( slength  > 0 ) {
       String temp = new String( buffer, offset,  slength );
       if ( !temp.trim().equals( "" ) )
       {try {
        int  start = tpd.getLength();
        tpd.insertString(start,
        temp + '\n', cstyle[depth]);
        int length = tpd.getLength()-start;
        tpd.setParagraphAttributes(start, length, cstyle[depth],true);
       }
      catch (BadLocationException ble) {
         System.err.println("Couldn't insert text.");
      }      }    }   }
```