Context

- XML is being used increasingly as an interchange format.
- This requires documents be processed algorithmically by programs. Two Application Program Interfaces (APIs) exist in Java to support this process
  - The Simple API for XML (SAX)
  - The Document Object Model (DOM) API
- While the 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
  - 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

Selected Java Classes Supporting Documents

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

JTextComponent

- JTextComponent is a subclass of JComponent
- JTextComponent is an abstract class
- It provides properties such as:
  - Read, write, or update
  - Cut, copy and paste
  - Select and replace ranges of text
  - Mapping keys to particular functions
- The three primary subclasses are:
  - JTextField
  - JTextArea
  - JEditorPane
### JTextField and JTextArea

- JTextField has a number of additional subclasses – i.e. JPasswordField, JFormattedTextFiled
  - JPasswordField provides visual security for entered data
  - JFormattedTextFiled allows for input data verification
- A JTextArea provides a light weight component for multi-line text.
  - The component can be sized in terms of rows and columns and it provides a scrollable interface.
  - 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 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 a subclass of the JEditorPane
- It provides all the basic capabilities needed to define a full featured word processor
- It allows for graphical and other components 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
Data Structures

- Several data structures support the use of structured documents in Java. They include:
  - Document
  - StylizedDocument
  - Style
  - JTree

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.

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.
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.

JTree

- A generalized tree structure
- Support for adding and removing nodes in a variety of ways:
  - Programmatically
  - Drag and Drop
- Built in support for multiple modes of rendering the tree and for collapsing or expanding nodes

Some Code Snippets

- 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
The Client GUI

The Server

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

**Sending a message**

```java
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**

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

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();
        }else if (localn.getNodeType()==Node.TEXT_NODE){
            NodeList lnl = localn.getChildNodes();
        }if (lnl.getLength()>0)
            {insertchildren(lnl, setbase(base, i),cn);}
    } //for
} //insertchildren method

public void startElement( String uri, String eleName,
                String raw, Attributes attributes ) throws SAXException
{ depth++;
    try {
        int start = tpd.getLength();
        tpd.insertString(start,"<"+eleName+">","",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
    }
}
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.");
            }
        } }
    }