core WEB programming

Simple API for XML

SAX

Agenda

- Introduction to SAX
- Installation and setup
- Steps for SAX parsing
- Defining a content handler
- Examples
  - Printing the Outline of an XML Document
  - Counting Book Orders
- Defining an error handler
- Validating a document

Simple API for XML (SAX)

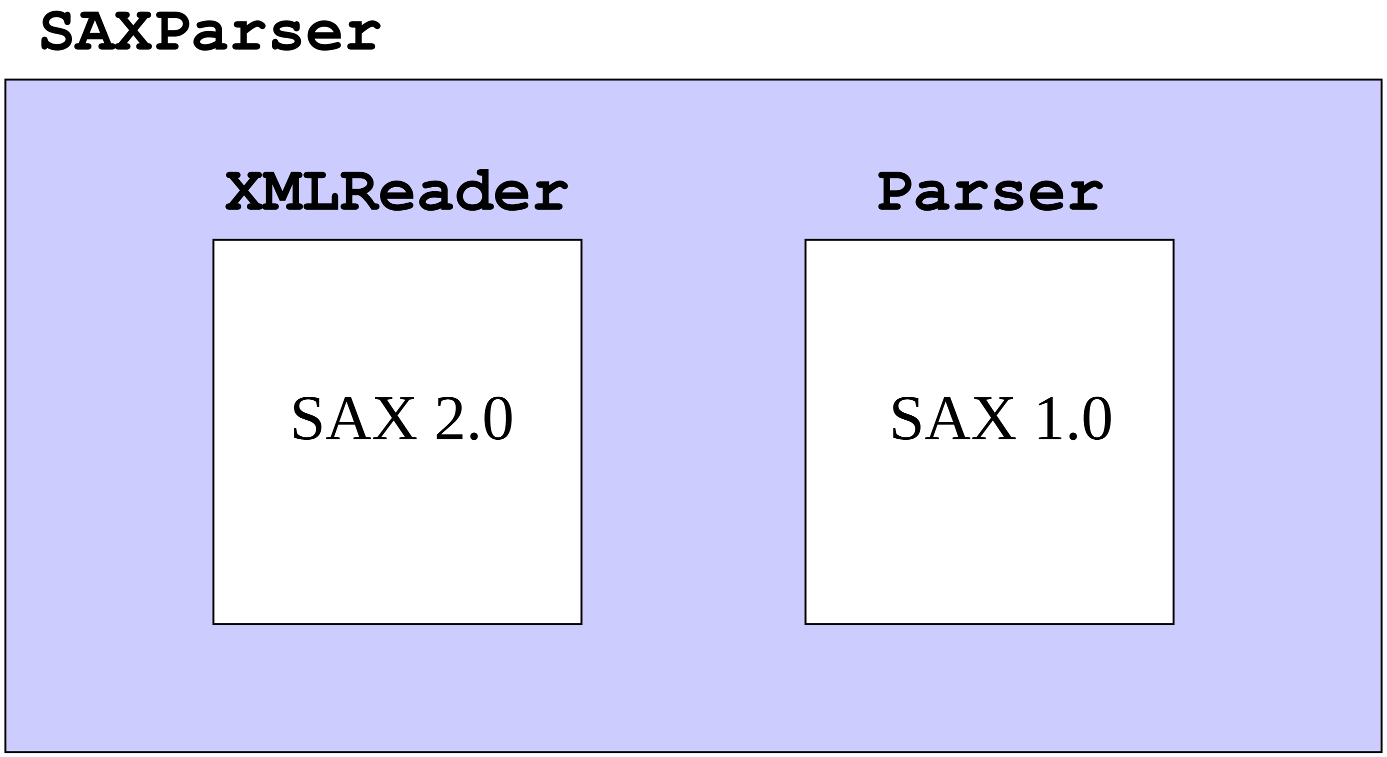
- **Parse and process XML documents**
- **Documents are read sequentially and callbacks are made to handlers**
- **Event-driven model for processing XML content**
- **SAX Versions**
  - SAX 1.0 (May 1998)
  - SAX 2.0 (May 2000)
    - Namespace addition
  - Official Website for SAX
    - `http://sax.sourceforge.net/`

SAX Advantages and Disadvantages

- **Advantages**
  - Do not need to process and store the entire document (low memory requirement)
    - Can quickly skip over parts not of interest
  - Fast processing
- **Disadvantages**
  - Limited API
    - Every element is processed through the same event handler
    - Need to keep track of location in document and, in cases, store temporary data
  - Only traverse the document once

Java API for XML Parsing (JAXP)

- **JAXP provides a vendor-neutral interface to the underlying SAX 1.0/2.0 parser**



SAX Installation and Setup (JDK 1.4)

- **All the necessary classes for SAX and JAXP are included with JDK 1.4**
  - See `javax.xml.*` packages
- **For SAX and JAXP with JDK 1.3 see following viewgraphs**

SAX Installation and Setup (JDK 1.3)

1. **Download a SAX 2-compliant parser**
   - Java-based XML parsers at `http://www.xml.com/pub/rg/Java_Parsers`
   - Recommend Apache Xerces-J parser at `http://xml.apache.org/xerces-j/`
2. **Download the Java API for XML Processing (JAXP)**
   - JAXP is a small layer on top of SAX which supports specifying parsers through system properties versus hard coded
   - See `http://java.sun.com/xml/`
   - Note: Apache Xerces-J already incorporates JAXP

SAX Installation and Setup (continued)

3. **Set your `CLASSPATH` to include the SAX (and JAXP) classes**

```
set CLASSPATH=xerces_install_dir\xerces.jar;
                           %CLASSPATH%
```

or

```
setenv CLASSPATH xerces_install_dir/xerces.jar:
                           $CLASSPATH
```

- For servlets, place `xerces.jar` in the server's `lib` directory
  - Note: Tomcat 4.0 is prebundled with `xerces.jar`
- Xerces-J already incorporates JAXP
  - For other parsers you may need to add `jaxp.jar` to your classpath and servlet `lib` directory

SAX Parsing

- **SAX parsing has two high-level tasks:**

  1. Creating a content handler to process the XML elements when they are encountered
  2. Invoking a parser with the designated content handler and document

Steps for SAX Parsing

1. Tell the system which parser you want to use
2. Create a parser instance
3. Create a content handler to respond to parsing events
4. Invoke the parser with the designated content handler and document

Step 1: Specifying a Parser

- **Approaches to specify a parser**
  - Set a system property for `javax.xml.parsers.SAXParserFactory`
  - Specify the parser in `jre_dir/lib/jaxp.properties`
  - Through the J2EE Services API and the class specified in `META-INF/services/javax.xml.parsers.SAXParserFactory`
  - Use system-dependant default parser (check documentation)

Specifying a Parser, Example

- **The following example:**
  - Permits the user to specify the parser through the command line `-D` option

```
java -Djavax.xml.parser.SAXParserFactory=
     com.sun.xml.parser.SAXParserFactoryImpl ...
```

  - Uses the Apache Xerces parser otherwise

```
public static void main(String[] args) {
  String jaxpPropertyName =
    "javax.xml.parsers.SAXParserFactory";
  if (System.getProperty(jaxpPropertyName) == null) {
    String apacheXercesPropertyValue =
      "org.apache.xerces.jaxp.SAXParserFactoryImpl";
    System.setProperty(jaxpPropertyName,
                       apacheXercesPropertyValue);
  }
  ...
}
```

Step 2: Creating a Parser Instance

- **First create an instance of a parser factory, then use that to create a `SAXParser` object**

```
SAXParserFactory factory =
  SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
```

  – To set up namespace awareness and validation, use

```
factory.setNamespaceAware(true)
factory.setValidating(true)
```

Step 3: Create a Content Handler

- **Content handler responds to parsing events**
  - Typically a subclass of `DefaultHandler`

```
public class MyHandler extends DefaultHandler {
  // Callback methods
  ...
}
```

- **Primary event methods (callbacks)**
  - startDocument, endDocument
    - Respond to the start and end of the document
  - startElement, endElement
    - Respond to the start and end tags of an element
  - characters, ignoreableWhitespace
    - Respond to the tag body

ContentHandler startElement Method

- **Declaration**

```
public void startElement(String nameSpaceURI,
                         String localName,
                         String qualifiedName,
                         Attributes attributes)
          throws SAXException
```

- **Arguments**
  - namespaceUri
    - URI uniquely identifying the namespace
  - localname
    - Element name without prefix
  - qualifiedName
    - Complete element name, including prefix
  - attributes
    - `Attributes` object representing the attributes of the element

Anatomy of an Element

```
                                     namespaceUri

<cwp:book xmlns:cwp="http://www.corewebprograming.com/xml/">

     qualifiedName                attribute[1]

  <cwp:chapter number="23" part="Server-side Programming">
    <cwp:title>XML Processing with Java</cwp:title>
  </cwp:chapter>

                   localname

</cwp:book>
```

ContentHandler characters Method

- **Declaration**

```
public void characters(char[] chars,
                       int startIndex,
                       int length)
           throws SAXException
```

- **Arguments**
  - chars
    - Relevant characters form XML document
    - To optimize parsers, the chars array may represent more of the XML document than just the element
    - **PCDATA** may cause multiple invocations of characters
  - startIndex
    - Starting position of element
  - length
    - The number of characters to extract

Step 4: Invoke the Parser

- Call the parse method, supplying:
  1. The content handler
  2. The XML document
     - File, input stream, or `org.xml.sax.InputSource`

```
parser.parse(filename, handler)
```

SAX Example 1: Printing the Outline of an XML Document

- **Approach**
  - Define a content handler to respond to three parts of an XML document: start tags, end tag, and tag bodies
  - Content handler implementation overrides the following three methods:
    - startElement
      - Prints a message when start tag is found with attributes listed in parentheses
      - Adjusts (increases by 2 spaces) the indentation
    - endElement
      - Subtracts 2 from the indentation and prints a message indicating that an end tag was found
    - characters
      - Prints the first word of the tag body

SAX Example 1: PrintHandler

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.StringTokenizer;

public class PrintHandler extends DefaultHandler {
  private int indentation = 0;

  /** When you see a start tag, print it out and then
   *  increase indentation by two spaces. If the
   *  element has attributes, place them in parens
   *  after the element name.
   */
  public void startElement(String namespaceUri,
                           String localName,
                           String qualifiedName,
                           Attributes attributes)
      throws SAXException {
    indent(indentation);
    System.out.print("Start tag: " + qualifiedName);
```

SAX Example 1: PrintHandler (continued)

```
    ...
    int numAttributes = attributes.getLength();
    // For <someTag> just print out "someTag". But for
    // <someTag att1="Val1" att2="Val2">, print out
    // "someTag (att1=Val1, att2=Val2).
    if (numAttributes > 0) {
      System.out.print(" (");
      for(int i=0; i<numAttributes; i++) {
        if (i>0) {
          System.out.print(", ");
        }
        System.out.print(attributes.getQName(i) + "=" +
                         attributes.getValue(i));
      }
      System.out.print(")");
    }
    System.out.println();
    indentation = indentation + 2;
  }
  ...
```

SAX Example 1: PrintHandler (continued)

```
/** When you see the end tag, print it out and decrease
 *  indentation level by 2.
 */

public void endElement(String namespaceUri,
                       String localName,
                       String qualifiedName)
    throws SAXException {
  indentation = indentation - 2;
  indent(indentation);
  System.out.println("End tag: " + qualifiedName);
}

private void indent(int indentation) {
  for(int i=0; i<indentation; i++) {
    System.out.print(" ");
  }
}
...
```

SAX Example 1: PrintHandler (continued)

```
  /** Print out the first word of each tag body. */

  public void characters(char[] chars,
                         int startIndex,
                         int length) {
    String data = new String(chars, startIndex, length);
    // Whitespace makes up default StringTokenizer delimeters
    StringTokenizer tok = new StringTokenizer(data);
    if (tok.hasMoreTokens()) {
      indent(indentation);
      System.out.print(tok.nextToken());
      if (tok.hasMoreTokens()) {
        System.out.println("...");
      } else {
        System.out.println();
      }
    }
  }
}
```

SAX Example 1: SAXPrinter

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SAXPrinter {
  public static void main(String[] args) {
    String jaxpPropertyName =
      "javax.xml.parsers.SAXParserFactory";
    // Pass the parser factory in on the command line with
    // -D to override the use of the Apache parser.
    if (System.getProperty(jaxpPropertyName) == null) {
      String apacheXercesPropertyValue =
        "org.apache.xerces.jaxp.SAXParserFactoryImpl";
      System.setProperty(jaxpPropertyName,
                         apacheXercesPropertyValue);
    }
```

SAX Example 1: SAXPrinter (continued)

```
    ...
    String filename;
    if (args.length > 0) {
      filename = args[0];
    } else {
      String[] extensions = { "xml", "tld" };
      WindowUtilities.setNativeLookAndFeel();
      filename =
        ExtensionFileFilter.getFileName(".", "XML Files",
                                          extensions);
      if (filename == null) {
        filename = "test.xml";
      }
    }
    printOutline(filename);
    System.exit(0);
  }
  ...
```

SAX Example 1: SAXPrinter (continued)

```
  ...
  public static void printOutline(String filename) {
    DefaultHandler handler = new PrintHandler();
    SAXParserFactory factory =
      SAXParserFactory.newInstance();
    try {
      SAXParser parser = factory.newSAXParser();
      parser.parse(filename, handler);
    } catch(Exception e) {
      String errorMessage =
        "Error parsing " + filename + ": " + e;
      System.err.println(errorMessage);
      e.printStackTrace();
    }
  }
}
```

SAX Example 1: orders.xml

```
<?xml version="1.0"?>
<orders>
  <order>
    <count>1</count>
    <price>9.95</price>
    <yacht>
      <manufacturer>Luxury Yachts, Inc.</manufacturer>
      <model>M-1</model>
      <standardFeatures oars="plastic"
                        lifeVests="none">
        false
      </standardFeatures>
    </yacht>
  </order>
  ...
</orders>
```

SAX Example 1: Result

```
Start tag: orders
  Start tag: order
    Start tag: count
      1
    End tag: count
    Start tag: price
      9.95
    End tag: price
    Start tag: yacht
      Start tag: manufacturer
        Luxury...
      End tag: manufacturer
      Start tag: model
        M-1
      End tag: model
      Start tag: standardFeatures (oars=plastic, lifeVests=none)
        false
      End tag: standardFeatures
    End tag: yacht
  End tag: order
  ...
End tag: orders
```

SAX Example 2: Counting Book Orders

- **Objective**
  - To process XML files that look like:

```
<orders>
  ...
  <count>23</count>
  <book>
    <isbn>013897930</isbn>
    ...
  </book>
  ...
</orders>
```

and count up how many copies of Core Web Programming (ISBN 013897930) are contained in the order

SAX Example 2: Counting Book Orders (continued)

- **Problem**
  - SAX doesn’t store data automatically
  - The `isbn` element comes after the `count` element
  - Need to record every count temporarily, but only add the temporary value (to the running total) when the ISBN number matches

SAX Example 2: Approach

- **Define a content handler to override the following four methods:**
  - startElement
    - Checks whether the name of the element is either `count` or `isbn`
    - Set flag to tell `characters` method be on the lookout
  - endElement
    - Again, checks whether the name of the element is either `count` or `isbn`
    - If so, turns off the flag that the `characters` method watches

SAX Example 2: Approach (continued)

- characters
  - Subtracts 2 from the indentation and prints a message indicating that an end tag was found
- endDocument
  - Prints out the running count in a Message Dialog

SAX Example 2: CountHandler

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

public class CountHandler extends DefaultHandler {
  private boolean collectCount = false;
  private boolean collectISBN = false;
  private int currentCount = 0;
  private int totalCount = 0;

  public void startElement(String namespaceUri,
                           String localName,
                           String qualifiedName,
                           Attributes attributes)
      throws SAXException {
    if (qualifiedName.equals("count")) {
      collectCount = true;
      currentCount = 0;
    } else if (qualifiedName.equals("isbn")) {
      collectISBN = true;
    }
  }
  ...
```

SAX Example 2: CountHandler (continued)

```
...
public void endElement(String namespaceUri,
                       String localName,
                       String qualifiedName)
    throws SAXException {
  if (qualifiedName.equals("count")) {
    collectCount = false;
  } else if (qualifiedName.equals("isbn")) {
    collectISBN = false;
  }
}

public void endDocument() throws SAXException {
  String message =
    "You ordered " + totalCount + " copies of \n" +
    "Core Web Programming Second Edition.\n";
  if (totalCount < 250) {
    message = message + "Please order more next time!";
  } else {
    message = message + "Thanks for your order.";
  }
  JOptionPane.showMessageDialog(null, message);
}
```

SAX Example 2: CountHandler (continued)

```
  ...
  public void characters(char[] chars, int startIndex,
                         int length) {
    if (collectCount || collectISBN) {
      String dataString =
        new String(chars, startIndex, length).trim();
      if (collectCount) {
        try {
          currentCount = Integer.parseInt(dataString);
        } catch(NumberFormatException nfe) {
          System.err.println("Ignoring malformed count: " +
                             dataString);
        }
      } else if (collectISBN) {
        if (dataString.equals("0130897930")) {
          totalCount = totalCount + currentCount;
        }
      }
    }
  }
}
```

SAX Example 2: CountBooks

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class CountBooks {
  public static void main(String[] args) {
    String jaxpPropertyName = "javax.xml.parsers.SAXParserFactory";
    // Use -D to override the use of the Apache parser.
    if (System.getProperty(jaxpPropertyName) == null) {
      String apacheXercesPropertyValue =
        "org.apache.xerces.jaxp.SAXParserFactoryImpl";
      System.setProperty(jaxpPropertyName,
                         apacheXercesPropertyValue);
    }
    String filename;
    if (args.length > 0) {
      filename = args[0];
    } else {
      ...
    }
    countBooks(filename);
    System.exit(0);
  }
```

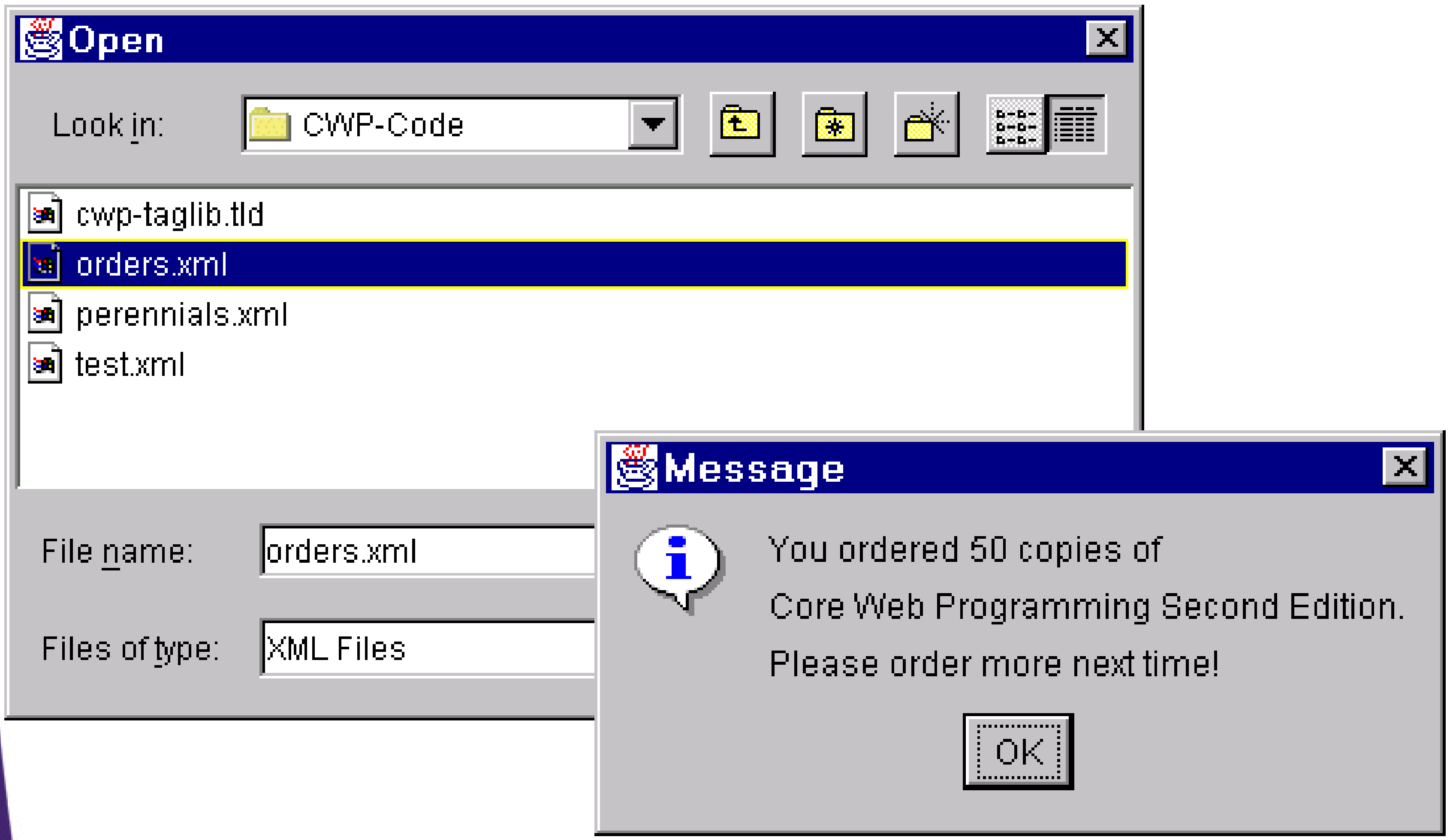
SAX Example 2: CountBooks (continued)

```
  private static void countBooks(String filename) {
    DefaultHandler handler = new CountHandler();
    SAXParserFactory factory =
      SAXParserFactory.newInstance();
    try {
      SAXParser parser = factory.newSAXParser();
      parser.parse(filename, handler);
    } catch(Exception e) {
      String errorMessage =
        "Error parsing " + filename + ": " + e;
      System.err.println(errorMessage);
      e.printStackTrace();
    }
  }
}
```

SAX Example 2: orders.xml

```
<?xml version="1.0"?>
<orders>
  <order>
    <count>37</count>
    <price>49.99</price>
    <book>
      <isbn>0130897930</isbn>
      <title>Core Web Programming Second Edition</title>
      <authors>
        <author>Marty Hall</author>
        <author>Larry Brown</author>
      </authors>
    </book>
  </order>
  ...
</orders>
```

SAX Example 2: Result



You ordered 50 copies of
Core Web Programming Second Edition.
Please order more next time!

Error Handlers

- Responds to parsing errors
  - Typically a subclass of `DefaultErrorHandler`
- Useful callback methods
  - error
    - Nonfatal error
    - Usual a result of document validity problems
  - fatalError
    - A fatal error resulting from a malformed document
  - Receive a `SAXParseException` from which to obtain the location of the problem (`getColumnNumber`, `getLineNumber`)

Error Handler Example

```
import org.xml.sax.*;
import org.apache.xml.utils.*;

class MyErrorHandler extends DefaultErrorHandler  {

  public void error(SAXParseException exception)
    throws SAXException {

    System.out.println(
      "**Parsing Error**\n" +
      "  Line:     " + exception.getLineNumber() + "\n" +
      "  URI:      " + exception.getSystemId() + "\n" +
      "  Message: " + exception.getMessage() + "\n");
    throw new SAXException("Error encountered");
  }
}
```

Namespace Awareness and Validation

- **Approaches**

1. Through the SAXParserFactory

```
factory.setNamespaceAware(true)
factory.setValidating(true)
SAXParser parser = factory.newSAXParser();
```

2. By setting XMLReader features

```
XMLReader reader = parser.getXMLReader();
reader.setFeature(
  "http://xml.org/sax/features/validation", true);
reader.setFeature(
  "http://xml.org/sax/features/namespaces", false);
```

  - Note: a `SAXParser` is a vendor-neutral wrapper around a SAX 2 `XMLReader`

Validation Example

```
public class SAXValidator {
  public static void main(String[] args) {
    String jaxpPropertyName =
   "javax.xml.parsers.SAXParserFactory";
    // Use -D to override the use of the Apache parser.
    if (System.getProperty(jaxpPropertyName) == null) {
      String apacheXercesPropertyValue =
        "org.apache.xerces.jaxp.SAXParserFactoryImpl";
      System.setProperty(jaxpPropertyName,
                         apacheXercesPropertyValue);
    }
    String filename;
    if (args.length > 0) {
      filename = args[0];
    } else {
      ...
    }
    validate(filename);
    System.exit(0);
  }
```

Validation Example (continued)

```
  ...
  public static void validate(String filename) {
    DefaultHandler contentHandler = new DefaultHandler();
    ErrorHandler errHandler = new MyErrorHandler();
    SAXParserFactory factory =
      SAXParserFactory.newInstance();
    factory.setValidating(true);
    try {
      SAXParser parser = factory.newSAXParser();
      XMLReader reader = parser.getXMLReader();
      reader.setContentHandler(contentHandler);
      reader.setErrorHandler(errHandler);
      reader.parse(new InputSource(filename));
    } catch(Exception e) {
      String errorMessage =
        "Error parsing " + filename;
      System.out.println(errorMessage);
    }
  }
}
```

Instructors.xml

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE jhu [
<!ELEMENT jhu (instructor)*>
<!ELEMENT instructor (firstname, lastname)+>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
]>
<jhu>
  <instructor>
    <firstname>Larry</firstname>
    <lastname>Brown</lastname>
  </instructor>
  <instructor>
    <lastname>Hall</lastname>
    <firstname>Marty</firstname>
  </instructor>
</jhu>
```

Validation Results

```
>java SAXValidator

Parsing Error:
  Line: 16
  URI:  file:///C:/CWP2-Book/chapter23/Instructors.xml
  Message: The content of element type "instructor‟
           must match "(firstname,lastname)+".

Error parsing C:\CWP2-Book\chapter23\Instructors.xml
```

Summary

- SAX processing of XML documents is fast and memory efficient
- JAXP is a simple API to provide vendor neutral SAX parsing
  - Parser is specified through system properties
- Processing is achieved through event call backs
  - Parser communicates with a DocumentHandler
  - May require tracking the location in document and storing data in temporary variables
- Parsing properties (validation, namespace awareness) are set through the SAXParser or underlying XMLReader

core WEB programming

Questions?