```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- excerpted from http://www.w3schools.com/xquery/xquery_example.asp -->
<bib>
        <book year="1994">
                <title>TCP/IP Illustrated</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
                <publisher>Addison-Wesley</publisher>
                <price>65.95</price>
        </book>

        <book year="1992">
                <title>Advanced Programming in the Unix environment</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
                <publisher>Addison-Wesley</publisher>
                <price>65.95</price>
        </book>

        <book year="2000">
                <title>Data on the Web</title>
                <author>
                        <last>Abiteboul</last>
                        <first>Serge</first>
                </author>
                <author>
                        <last>Buneman</last>
                        <first>Peter</first>
                </author>
                <author>
                        <last>Suciu</last>
                        <first>Dan</first>
                </author>
                <publisher>Morgan Kaufmann Publishers</publisher>
                <price>39.95</price>
        </book>

        <book year="1999">
                <title>The Technology and Content for Digital TV</title>
                <editor>
                        <last>Gerbarg</last>
                        <first>Darcy</first>
                        <affiliation>CITI</affiliation>
                </editor>
                <publisher>Kluwer Academic Publishers</publisher>
                <price>129.95</price>
        </book>
</bib>

```

```
<!ELEMENT  emails  (message+)>
<!ELEMENT  message  (subject?, body, reply*)>
  <!ATTLIST  message
    num   ID     #REQUIRED
    to    CDATA  #REQUIRED
    from  CDATA  #FIXED     "brenda&#64;xyzcompany.com"
    date  CDATA  #REQUIRED>
<!ELEMENT  subject  EMPTY>
  <!ATTLIST   subject
    title  CDATA  #IMPLIED>
<!ELEMENT  body   ANY>
<!ELEMENT  reply  EMPTY>
  <!ATTLIST  reply
    status  (yes | no)  "no">
```

```xml
<?xml version="1.0" standalone="no"?>
<!-- from http://wps.aw.com/aw_webwizard/0,6065,184738-,00.html -->
<!DOCTYPE  emails SYSTEM "emails.dtd">
<emails>
  <message num="a1"
    to="joe&#64;acmeshipping.com"
    from="brenda&#64;xyzcompany.com"
    date="02/09/01">
  <subject title="Order 10011" />
  <body>
    Joe,
      Please let me know if order number 10011 has shipped.
    Thanks,
    Brenda
  </body>
  <reply status="yes"/>
  </message>
</emails>
```

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE foo [
  <!ELEMENT foo (bar,baz)>
  <!ELEMENT bar (#PCDATA)>
  <!ELEMENT baz (#PCDATA)>
]>

<!-- notice that foo contains ignorable whitespace -->

<foo>
  <bar/>
  <baz/>
</foo>
```

```xml
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE foo [
  <!ELEMENT foo (bar,baz)>
  <!ELEMENT bar EMPTY>
  <!ELEMENT baz EMPTY>
]>

<!-- notice that the ordering of baz and bar is invalid! -->

<foo>
  <baz/>
  <bar/>
</foo>
```

```java
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXParseException;


/**
 * Lecture 8’s demonstration of validation.
 *
 * @author  Computer Science E-259
 **/

public class SAXValidator extends DefaultHandler
{
    /**
     * Main driver.  Expects one command-line argument: the name of the XML file to
     * validate
     *
     * @param argv [0] - filename
     */

    public static void main(String [] argv)
    {
        // grab filename
        String input = argv[0];

        try
        {
            // instantiate a SAX parser factory
            SAXParserFactory factory = SAXParserFactory.newInstance();

            // enable validation
            factory.setValidating(true);

            // instantiate a SAX parser
            SAXParser parser = factory.newSAXParser();

            // instantiate our little handler
            SAXValidator handler = new SAXValidator();

            // parse the file
            parser.parse(input, handler);

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }


    /**
     * Receive notification of a recoverable parser error.
     *
     * @param e  the exception thrown
     */

    public void error(SAXParseException e)
    {
        System.out.println("Parsing error at " + e.getLineNumber() +
                           ":" + e.getColumnNumber() +
                           ": " + e.getMessage());
    }


    /**
     * Receive notification of a parser warning.
     *
     * @param e  the exception thrown
     */

    public void warning(SAXParseException e)
    {
        System.out.println("Parsing warning at " + e.getLineNumber() +
                           ":" + e.getColumnNumber() +
                           ": " + e.getMessage());
    }


    /**
     * Report a fatal XML parsing error.
     *
     * @param e  the exception thrown
     */

    public void fatalError(SAXParseException e)
    {
        System.out.println("Fatal parsing warning at " + e.getLineNumber() +
                           ":" + e.getColumnNumber() +
                           ": " + e.getMessage());
    }
}
```

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ELEMENT bar EMPTY>
  <!ELEMENT baz EMPTY>
]>

<!-- notice that foo contains significant whitespace! -->

<foo>
  <bar/>
  <baz/>
</foo>
```

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE SONG [
 <!ELEMENT SONG (TITLE, COMPOSER+, PRODUCER*, PUBLISHER*, LENGTH?, YEAR?, ARTIST+)>
 <!ELEMENT TITLE (#PCDATA)>
 <!ELEMENT COMPOSER (#PCDATA)>
 <!ELEMENT PRODUCER (#PCDATA)>
 <!ELEMENT PUBLISHER (#PCDATA)>
 <!ELEMENT LENGTH (#PCDATA)>
 <!ELEMENT YEAR (#PCDATA)>
 <!ELEMENT ARTIST (#PCDATA)>
]>

<SONG>
  <TITLE>Everyday</TITLE>
  <COMPOSER>Dave</COMPOSER>
  <COMPOSER>Boyd Tinsley</COMPOSER>
  <PRODUCER>Dave Matthews</PRODUCER>
  <PUBLISHER>BMG</PUBLISHER>
  <LENGTH>12:20</LENGTH>
  <YEAR>2001</YEAR>
  <ARTIST>Dave Matthews Band</ARTIST>
</SONG>
```

```xml
<?xml version="1.0" standalone="no"?>

<!DOCTYPE SONG SYSTEM "song.dtd">

<SONG>
  <TITLE>Everyday</TITLE>
  <COMPOSER>Dave</COMPOSER>
  <COMPOSER>Boyd Tinsley</COMPOSER>
  <PRODUCER>Dave Matthews</PRODUCER>
  <PUBLISHER>BMG</PUBLISHER>
  <LENGTH>12:20</LENGTH>
  <YEAR>2001</YEAR>
  <ARTIST>Dave Matthews Band</ARTIST>
</SONG>
```

```
<!ELEMENT SONG (TITLE, COMPOSER+, PRODUCER*, PUBLISHER*, LENGTH?, YEAR?, ARTIST+)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT COMPOSER (#PCDATA)>
<!ELEMENT PRODUCER (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
<!ELEMENT LENGTH (#PCDATA)>
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT ARTIST (#PCDATA)>
```

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.AttributesImpl;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXParseException;


/**
 * Lecture 8’s demonstration of whitespace handling.
 *
 * @author  Computer Science E-259
 **/

public class WhitespaceDemo extends DefaultHandler
{
    /**
     * Main driver.  Expects one command-line argument:
     * the name of the file to parse.
     *
     * @param argv [0] - filename
     */

    public static void main(String [] argv)
    {
        // grab filename
        String input = argv[0];

        try
        {
            // instantiate a SAX parser factory
            SAXParserFactory factory = SAXParserFactory.newInstance();

            // enable validation
            factory.setValidating(true);

            // instantiate a SAX parser
            SAXParser parser = factory.newSAXParser();

            // instantiate our little handler
            WhitespaceDemo handler = new WhitespaceDemo();

            // parse the file
```

```
                parser.parse(input, handler);

            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }


        /**
         * Report a startElement event.
         *
         * @param   uri namespace
         * @param   localName   name of element, sans namespace
         * @param   qName       name of element, with namespace
         * @param   attributes  element’s collection of attributes
         *
         * @throws  SAXException    general SAX error or warning
         */

        public void startElement(String uri, String localName,
                                 String qName, Attributes atts)
         throws SAXException
        {
            System.out.print("startElement(\"" + qName + ", {");
            for (int i = 0; i < atts.getLength(); i++)
            {
                System.out.print("(\"" + atts.getQName(i) + "\", \"" +
                                 atts.getValue(i) + "\")");
                if (i != atts.getLength() - 1)
                    System.out.print(", ");
            }
            System.out.println("});");
        }


        /**
         * Report a characters event.
         *
         * @param   ch      characters
         * @param   start   start position in the character array
         * @param   length  number of characters to use from the character array
         *
         * @throws  SAXException    general SAX error or warning
```

```
     */

    public void characters(char[] ch, int start, int length)
     throws SAXException
    {
        System.out.println("characters(\"" + new String(ch, start, length) +
                           "\");");
    }


    /**
     * Report an endElement event.
     *
     * @param   uri         namespace
     * @param   localName   name of element, sans namespace
     * @param   qName       name of element, with namespace
     *
     * @throws  SAXException    general SAX error or warning
     */

    public void endElement(String uri, String localName, String qName)
     throws SAXException
    {
        System.out.println("endElement(\"" + qName + "\");");
    }


    /**
     * Report a startDocument event.
     */

    public void startDocument() throws SAXException
    {
        System.out.println("\nstartDocument();");
    }


    /**
     * Report an endDocument event.
     *
     * @throws  SAXException    general SAX error or warning
     */

    public void endDocument() throws SAXException
    {
        System.out.println("endDocument();\n");
    }


    /**
     * Receive notification of a recoverable parser error.
     *
     * @param e  the exception thrown
     */

    public void error (SAXParseException e)
    {
        System.out.println("Parsing error:  " + e.getMessage());
    }


    /**
     * Receive notification of a parser warning.
     *
     * @param e  the exception thrown
     */

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


    /**
     * Report a fatal XML parsing error.
     *
     * @param e  the exception thrown
     */

    public void fatalError (SAXParseException e)
    {
        System.out.println("Fatal parsing error:  " + e.getMessage());
        System.exit(1);
    }
}
```

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
    <head>
        <title/>
    </head>
    <body/>
</html>
```