XML Validation

Description

For this assignment, you will be defining the XML schema for an XML representation of shopping order data. For the convenience of data representation, the products in this order are all books. You will have to design your own **XML** representation, as well as the **XML schema** of your representation.

- The XML representation of **order** should contain **order id**, **description**, **request date**, and many **lines item**
- Each line item element should contain **book**, **quantity**, and **price**. Price (which is price per book) should be restricted to values greater than 0
- Each book should contain **book id**, **book name**, **genre**, **publish date**, and many **authors**. Genre should be restricted so that it may contain only the following values {science-fiction, mystery, thriller, drama}
- Each author should contain **bio**, **last name**, **first name**, and optional **pen name**.

Evaluation criteria

For the most part the evaluation will be based on the following tests that your schema:

- allows many **line items**
- allows many **authors** for a book and have an OPTIONAL **pen name**
- allows to specify **price** and restricts to values greater than 0
- allows to specify **genre** but restricts the list of values to be EXACTLY that state above. (EXACT so that it enables me to test everybody's schema easily.)

Note: What does it mean by MOST PART of the evaluation criteria, what is the other part? Answer: We wish to be tolerant to simple changes in your schema that make better sense to you. We also wish to provide you with the evaluation criteria up front to help you debug before you submit so that you can maximize your score. Normally, the MOST PART should be everything. However, we have to account for unforeseen cases that call for judgement. For example, a dramatic situation where some student submits a bizarre schema where an order has many line-items and line-items have many authors (instead of line-items having a book that has many authors). Even though such a bizarre schema will pass the stated evaluation criteria, points will be taken off.

DOMEcho

Before submitting your files, you should use DOMEcho validator to validate your XML and schema. Since different validators might have slightly different behaviors, please make sure your XML and schema can work using DOMEcho validator. We will need **Java** to execute DOMEcho.

```
javac DOMEcho.java
java DOMEcho -xsdss P3.xsd P3.xml
```

```
// JAXP packages
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

import java.io.*;


/**
 * This is a program to echo a DOM tree using DOM Level 2 interfaces.  Use
 * JAXP to load an XML file and create a DOM tree.  DOM currently does not
 * provide a method to do this.  (This is planned for Level 3.)  See the
 * method "main" for the three basic steps.  Once the application obtains a
 * DOM Document tree, it dumps out the nodes in the tree and associated
 * node attributes for each node.
 *
 * This program also shows how to validate a document along with using an
 * ErrorHandler to capture validation errors.
 *
 * Note: Program flags may be used to create non-conformant but possibly
 * useful DOM trees.  In some cases, particularly with element content
 * whitespace, applications may not want to rely on JAXP to filter out
 * these nodes but may want to skip the nodes themselves so the application
 * will be more robust.
 *
 * Update 2002-04-18: Added code that shows how to use JAXP 1.2 features to
 * support W3C XML Schema validation.  See the JAXP 1.2 maintenance review
 * specification for more information on these features.
 *
 * @author Edwin Goei
 */
public class DOMEcho {
    /** All output will use this encoding */
    static final String outputEncoding = "UTF-8";

    /** Output goes here */
    private PrintWriter out;

    /** Indent level */
    private int indent = 0;

    /** Indentation will be in multiples of basicIndent  */
    private final String basicIndent = "  ";

    /** Constants used for JAXP 1.2 */
    static final String JAXP_SCHEMA_LANGUAGE =
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
    static final String W3C_XML_SCHEMA =
        "http://www.w3.org/2001/XMLSchema";
    static final String JAXP_SCHEMA_SOURCE =
        "http://java.sun.com/xml/jaxp/properties/schemaSource";

    DOMEcho(PrintWriter out) {
        this.out = out;
    }

    /**
     * Echo common attributes of a DOM2 Node and terminate output with an
     * EOL character.
     */
    private void printlnCommon(Node n) {
        out.print(" nodeName=\"" + n.getNodeName() + "\"");

        String val = n.getNamespaceURI();
        if (val != null) {
            out.print(" uri=\"" + val + "\"");
        }

        val = n.getPrefix();
        if (val != null) {
            out.print(" pre=\"" + val + "\"");
        }

        val = n.getLocalName();
        if (val != null) {
            out.print(" local=\"" + val + "\"");
        }

        val = n.getNodeValue();
        if (val != null) {
            out.print(" nodeValue=");
            if (val.trim().equals("")) {
                // Whitespace
                out.print("[WS]");
            } else {
                out.print("\"" + n.getNodeValue() + "\"");
            }
        }
        out.println();
    }

    /**
     * Indent to the current level in multiples of basicIndent
     */
    private void outputIndentation() {
        for (int i = 0; i < indent; i++) {
            out.print(basicIndent);
        }
    }

    /**
     * Recursive routine to print out DOM tree nodes
     */
    private void echo(Node n) {
        // Indent to the current level before printing anything
```

```
outputIndentation();

int type = n.getNodeType();
switch (type) {
case Node.ATTRIBUTE_NODE:
    out.print("ATTR:");
    printlnCommon(n);
    break;
case Node.CDATA_SECTION_NODE:
    out.print("CDATA:");
    printlnCommon(n);
    break;
case Node.COMMENT_NODE:
    out.print("COMM:");
    printlnCommon(n);
    break;
case Node.DOCUMENT_FRAGMENT_NODE:
    out.print("DOC_FRAG:");
    printlnCommon(n);
    break;
case Node.DOCUMENT_NODE:
    out.print("DOC:");
    printlnCommon(n);
    break;
case Node.DOCUMENT_TYPE_NODE:
    out.print("DOC_TYPE:");
    printlnCommon(n);

    // Print entities if any
    NamedNodeMap nodeMap = ((DocumentType)n).getEntities();
    indent += 2;
    for (int i = 0; i < nodeMap.getLength(); i++) {
        Entity entity = (Entity)nodeMap.item(i);
        echo(entity);
    }
    indent -= 2;
    break;
case Node.ELEMENT_NODE:
    out.print("ELEM:");
    printlnCommon(n);

    // Print attributes if any.  Note: element attributes are not
    // children of ELEMENT_NODEs but are properties of their
    // associated ELEMENT_NODE.  For this reason, they are printed
    // with 2x the indent level to indicate this.
    NamedNodeMap atts = n.getAttributes();
    indent += 2;
    for (int i = 0; i < atts.getLength(); i++) {
        Node att = atts.item(i);
        echo(att);
    }
    indent -= 2;
    break;
case Node.ENTITY_NODE:
    out.print("ENT:");
    printlnCommon(n);
    break;
```

```
        case Node.ENTITY_REFERENCE_NODE:
            out.print("ENT_REF:");
            printlnCommon(n);
            break;
        case Node.NOTATION_NODE:
            out.print("NOTATION:");
            printlnCommon(n);
            break;
        case Node.PROCESSING_INSTRUCTION_NODE:
            out.print("PROC_INST:");
            printlnCommon(n);
            break;
        case Node.TEXT_NODE:
            out.print("TEXT:");
            printlnCommon(n);
            break;
        default:
            out.print("UNSUPPORTED NODE: " + type);
            printlnCommon(n);
            break;
        }

        // Print children if any
        indent++;
        for (Node child = n.getFirstChild(); child != null;
             child = child.getNextSibling()) {
            echo(child);
        }
        indent--;
    }

    private static void usage() {
        System.err.println("Usage: DOMEcho [-options] <file.xml>");
        System.err.println("       -dtd = DTD validation");
        System.err.println("       -xsd | -xsdss <file.xsd> = W3C XML Schema 
validation using xsi: hints");
        System.err.println("           in instance document or schema source 
<file.xsd>");
        System.err.println("       -ws = do not create element content whitespace 
nodes");
        System.err.println("       -co[mments] = do not create comment nodes");
        System.err.println("       -cd[ata] = put CDATA into Text nodes");
        System.err.println("       -e[ntity-ref] = create EntityReference 
nodes");
        System.err.println("       -usage or -help = this message");
        System.exit(1);
    }

    public static void main(String[] args) throws Exception {
        String filename = null;
        boolean dtdValidate = false;
        boolean xsdValidate = false;
        String schemaSource = null;

        boolean ignoreWhitespace = false;
        boolean ignoreComments = false;
        boolean putCDATAIntoText = false;
boolean createEntityRefs = false;

for (int i = 0; i < args.length; i++) {
    if (args[i].equals("-dtd")) {
        dtdValidate = true;
    } else if (args[i].equals("-xsd")) {
        xsdValidate = true;
    } else if (args[i].equals("-xsdss")) {
        if (i == args.length - 1) {
            usage();
        }
        xsdValidate = true;
        schemaSource = args[++i];
    } else if (args[i].equals("-ws")) {
        ignoreWhitespace = true;
    } else if (args[i].startsWith("-co")) {
        ignoreComments = true;
    } else if (args[i].startsWith("-cd")) {
        putCDATAIntoText = true;
    } else if (args[i].startsWith("-e")) {
        createEntityRefs = true;
    } else if (args[i].equals("-usage")) {
        usage();
    } else if (args[i].equals("-help")) {
        usage();
    } else {
        filename = args[i];

        // Must be last arg
        if (i != args.length - 1) {
            usage();
        }
    }
}
if (filename == null) {
    usage();
}

// Step 1: create a DocumentBuilderFactory and configure it
DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();

// Set namespaceAware to true to get a DOM Level 2 tree with nodes
// containing namesapce information.  This is necessary because the
// default value from JAXP 1.0 was defined to be false.
dbf.setNamespaceAware(true);

// Set the validation mode to either: no validation, DTD
// validation, or XSD validation
dbf.setValidating(dtdValidate || xsdValidate);
if (xsdValidate) {
    try {
        dbf.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
    } catch (IllegalArgumentException x) {
        // This can happen if the parser does not support JAXP 1.2
        System.err.println(
                    "Error: JAXP DocumentBuilderFactory attribute not
recognized: "
                    + JAXP_SCHEMA_LANGUAGE);
                System.err.println(
                    "Check to see if parser conforms to JAXP 1.2 spec.");
                System.exit(1);
            }
        }

        // Set the schema source, if any.  See the JAXP 1.2 maintenance
        // update specification for more complex usages of this feature.
        if (schemaSource != null) {
            dbf.setAttribute(JAXP_SCHEMA_SOURCE, new File(schemaSource));
        }

        // Optional: set various configuration options
        dbf.setIgnoringComments(ignoreComments);
        dbf.setIgnoringElementContentWhitespace(ignoreWhitespace);
        dbf.setCoalescing(putCDATAIntoText);
        // The opposite of creating entity ref nodes is expanding them inline
        dbf.setExpandEntityReferences(!createEntityRefs);

        // Step 2: create a DocumentBuilder that satisfies the constraints
        // specified by the DocumentBuilderFactory
        DocumentBuilder db = dbf.newDocumentBuilder();

        // Set an ErrorHandler before parsing
        OutputStreamWriter errorWriter =
            new OutputStreamWriter(System.err, outputEncoding);
        db.setErrorHandler(
            new MyErrorHandler(new PrintWriter(errorWriter, true)));

        // Step 3: parse the input file
        Document doc = db.parse(new File(filename));

        // Print out the DOM tree
        // Commented out by Munindar Singh
//         OutputStreamWriter outWriter =
//             new OutputStreamWriter(System.out, outputEncoding);
//         new DOMEcho(new PrintWriter(outWriter, true)).echo(doc);
    }

    // Error handler to report errors and warnings
    private static class MyErrorHandler implements ErrorHandler {
        /** Error handler output goes here */
        private PrintWriter out;

        MyErrorHandler(PrintWriter out) {
            this.out = out;
        }

        /**
         * Returns a string describing parse exception details
         */
        private String getParseExceptionInfo(SAXParseException spe) {
            String systemId = spe.getSystemId();
            if (systemId == null) {
                systemId = "null";
            }
            String info = "URI=" + systemId +
                " Line=" + spe.getLineNumber() +
                ": " + spe.getMessage();
            return info;
        }

        // The following methods are standard SAX ErrorHandler methods.
        // See SAX documentation for more info.

        public void warning(SAXParseException spe) throws SAXException {
            out.println("Warning: " + getParseExceptionInfo(spe));
        }

        public void error(SAXParseException spe) throws SAXException {
            String message = "Error: " + getParseExceptionInfo(spe);
           // Next two lines (insert and comment) modified by Munindar Singh
            out.println(message);
           //            throw new SAXException(message);
        }

        public void fatalError(SAXParseException spe) throws SAXException {
            String message = "Fatal Error: " + getParseExceptionInfo(spe);
           // Next two lines (insert and comment) modified by Munindar Singh
            out.println(message);
           //            throw new SAXException(message);
        }
    }
}
```

Deliverable

- Submit the **XML schema** and name it P3.xsd.
- Submit an **example XML file** named P3.xml that instantiates the schema.