# XMLとJava

# 構造化された文書

- 技術文書、設定ファイルなど
- 文書の構造、文書の内容、文書の表示を分離する
- 例：技術文書
  - 文書の構造：章、説、段落、箇条書きなど
  - 文書の内容
  - 文書の表示：章のタイトルのフォントサイズなど

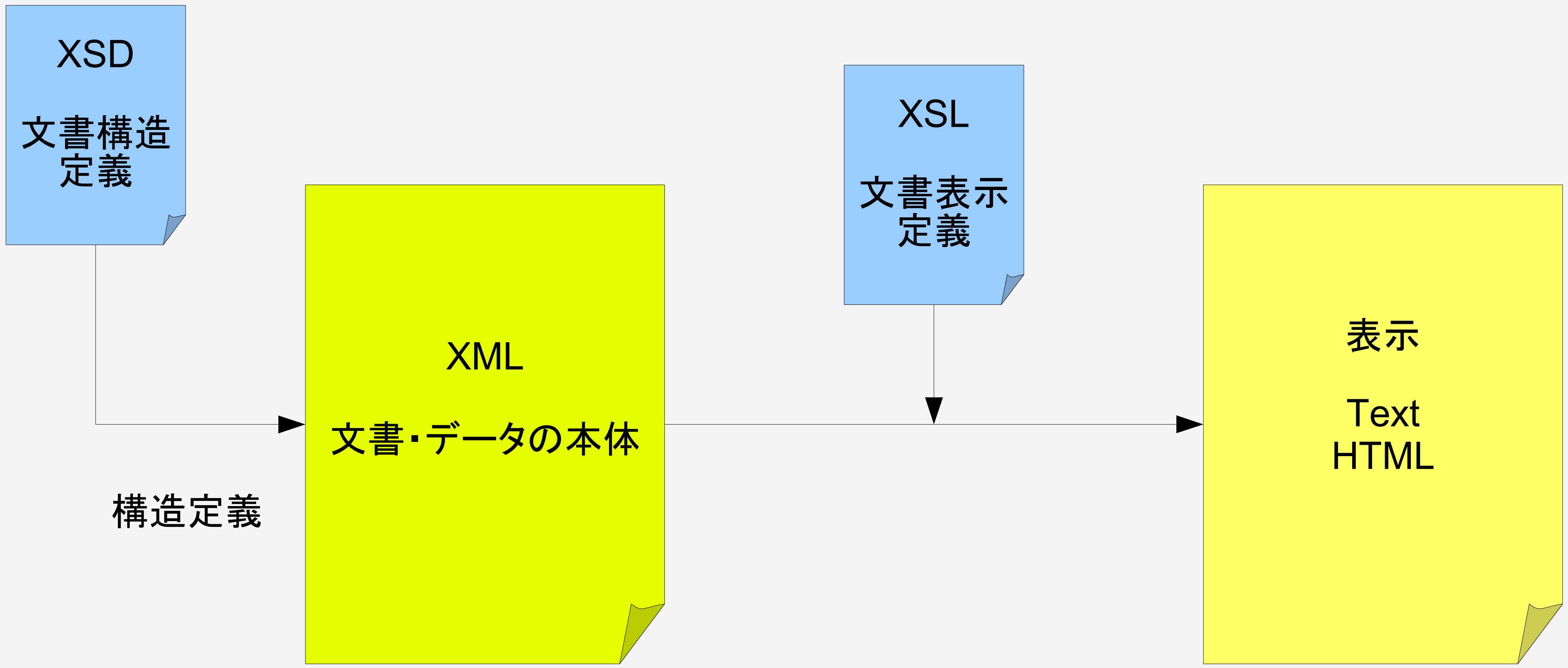
- 例：設定ファイル
  - 文書の構造：各設定項目の定義
  - 文書の内容：設定内容
  - 文書の表示：マニュアルに掲載する書式

# HTML

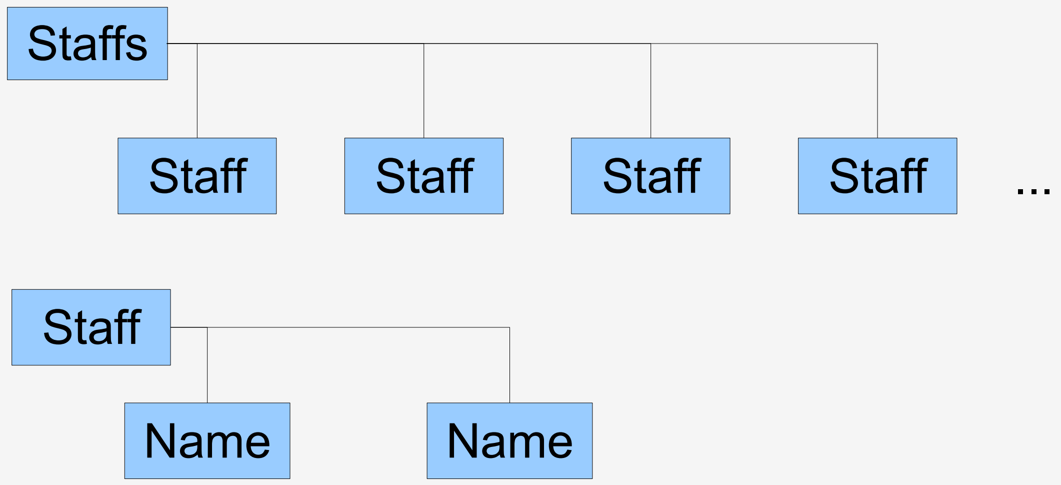
- Hyper Text Markup Language
- Web ページの記述言語
- 文書の構造、内容、表示を分離できる
  - 構造は仕様として定義され変更できない
  - タグを閉じ忘れても表示できる：ゆるいチェック
  - 表示をCSSとして分離できる：分離しなくてもよい

# XML

- eXtensible Markup Language
- 構造を別に定義する
  - XML SchemeまたはDTD (Document Type Definition)
- 表示を別に定義する
  - XSL (XML Stylesheet Language)


XSD
文書構造
定義
XSL
文書表示
定義
XML
文書・データの本体
構造定義
表示
Text
HTML

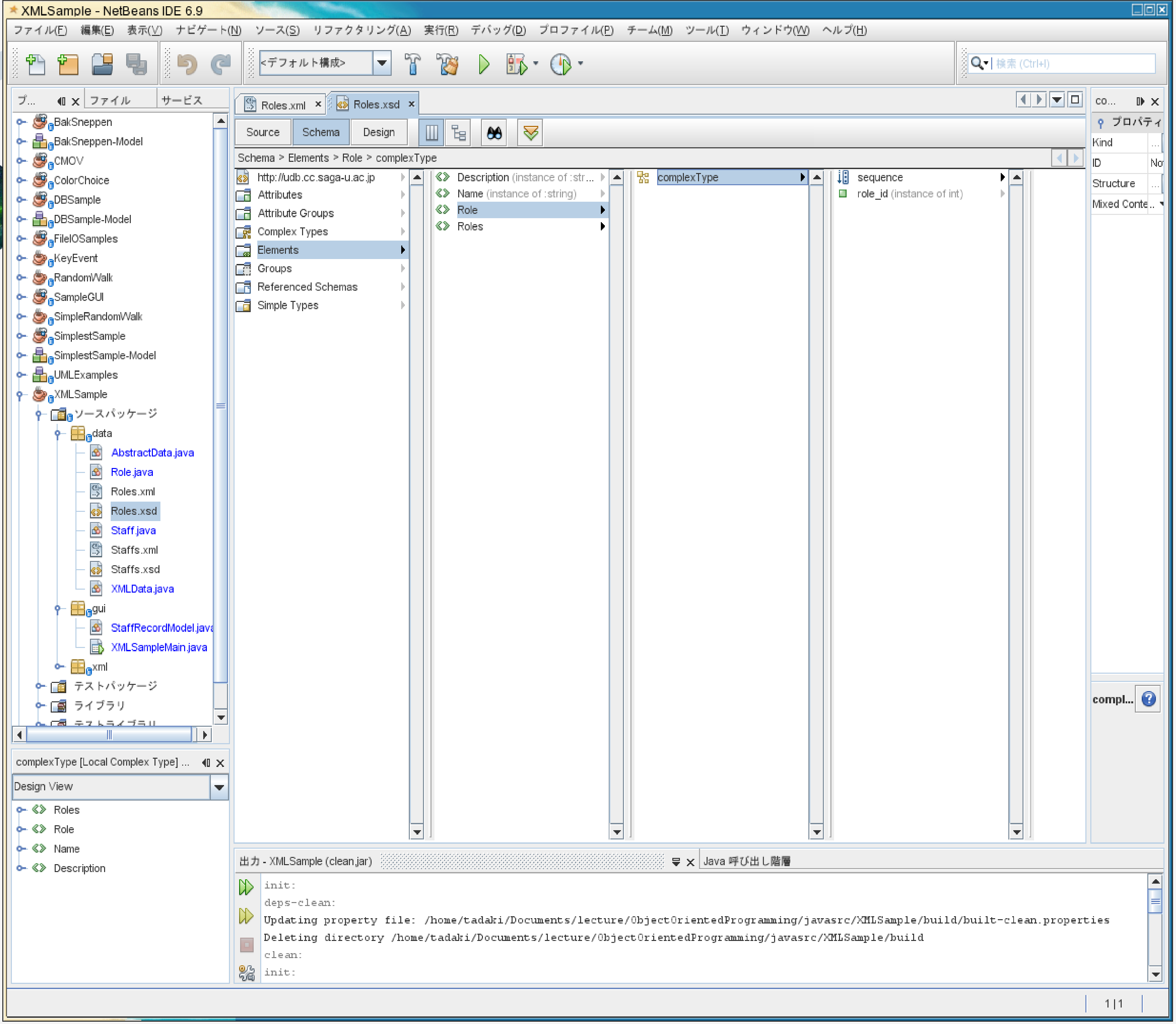
## Staffs.xsd



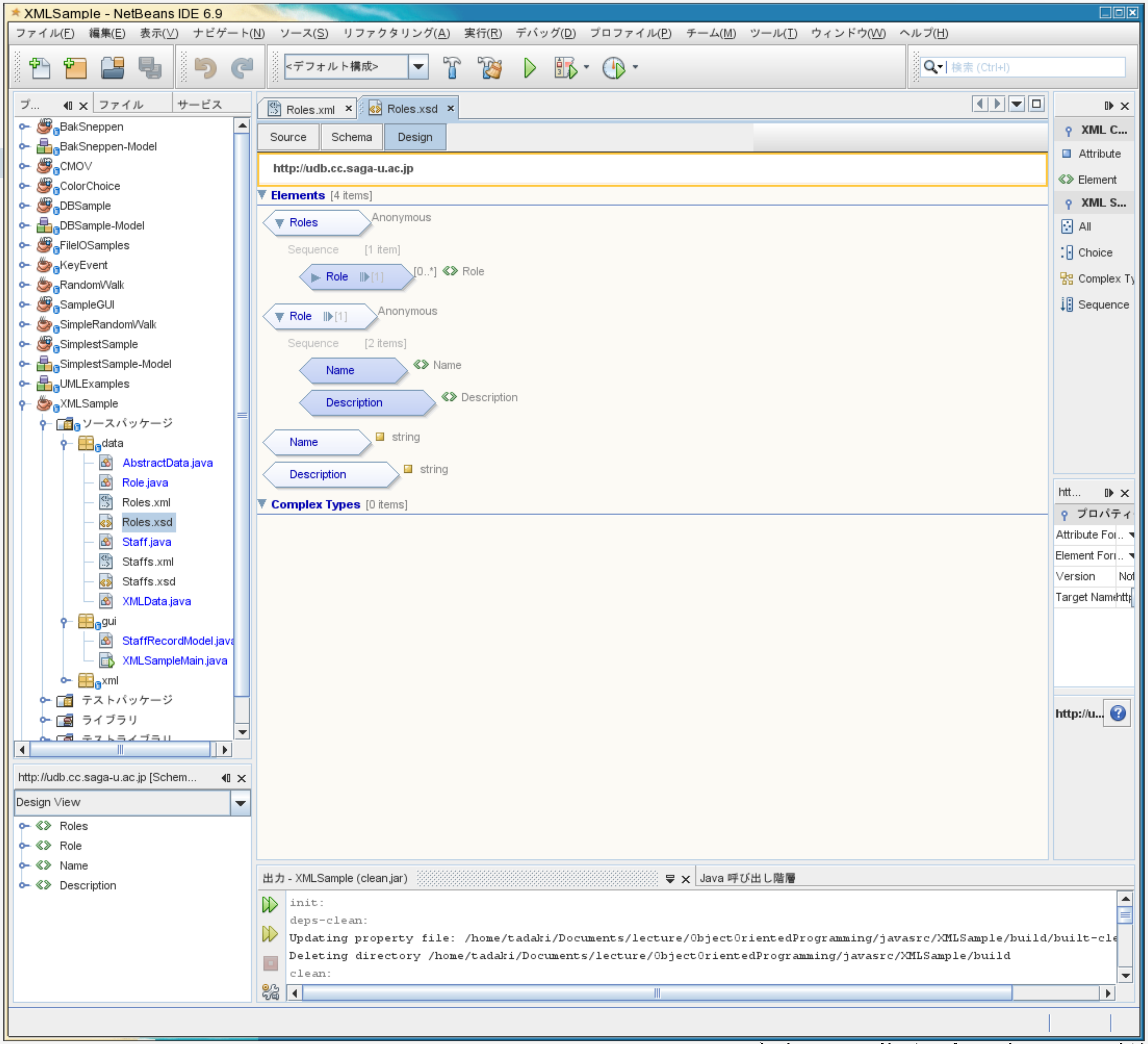
```xml
<?xml version="1.0" encoding="UTF-8"?>

<Staffs  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns='http://udb.cc.saga-u.ac.jp'
   xsi:schemaLocation='http://udb.cc.saga-u.ac.jp Staffs.xsd'>
    <Staff staff_id="1" valid="true" reg_date="2009-01-01T00:00:00" role="1">
        <Name>只木</Name>
        <Description></Description>
    </Staff>
    <Staff staff_id="2" valid="true" reg_date="2009-01-01T00:00:00" role="2">
        <Name>江藤</Name>
        <Description></Description>
    </Staff>
    <Staff staff_id="3" valid="true" reg_date="2009-01-01T00:00:00" role="3">
        <Name>渡辺</Name>
        <Description></Description>
    </Staff>
    <Staff staff_id="4" valid="true" reg_date="2009-01-01T00:00:00" role="4">
        <Name>大谷</Name>
        <Description></Description>
    </Staff>

</Staffs>
```

```
    <xsd:element name="Staff">        要素の定義
        <xsd:complexType>
            <xsd:sequence>                    子の要素
                <xsd:element ref="Name"/>
                <xsd:element ref="Description"/>
            </xsd:sequence>
            <xsd:attribute name="staff_id" type="xsd:int" use="required"/>        属性
            <xsd:attribute name="valid" type="xsd:boolean" use="required"/>
            <xsd:attribute name="reg_date" type="xsd:dateTime"
use="required"/>
            <xsd:attribute name="role" type="xsd:int" use="required"/>
        </xsd:complexType>
    </xsd:element>
```

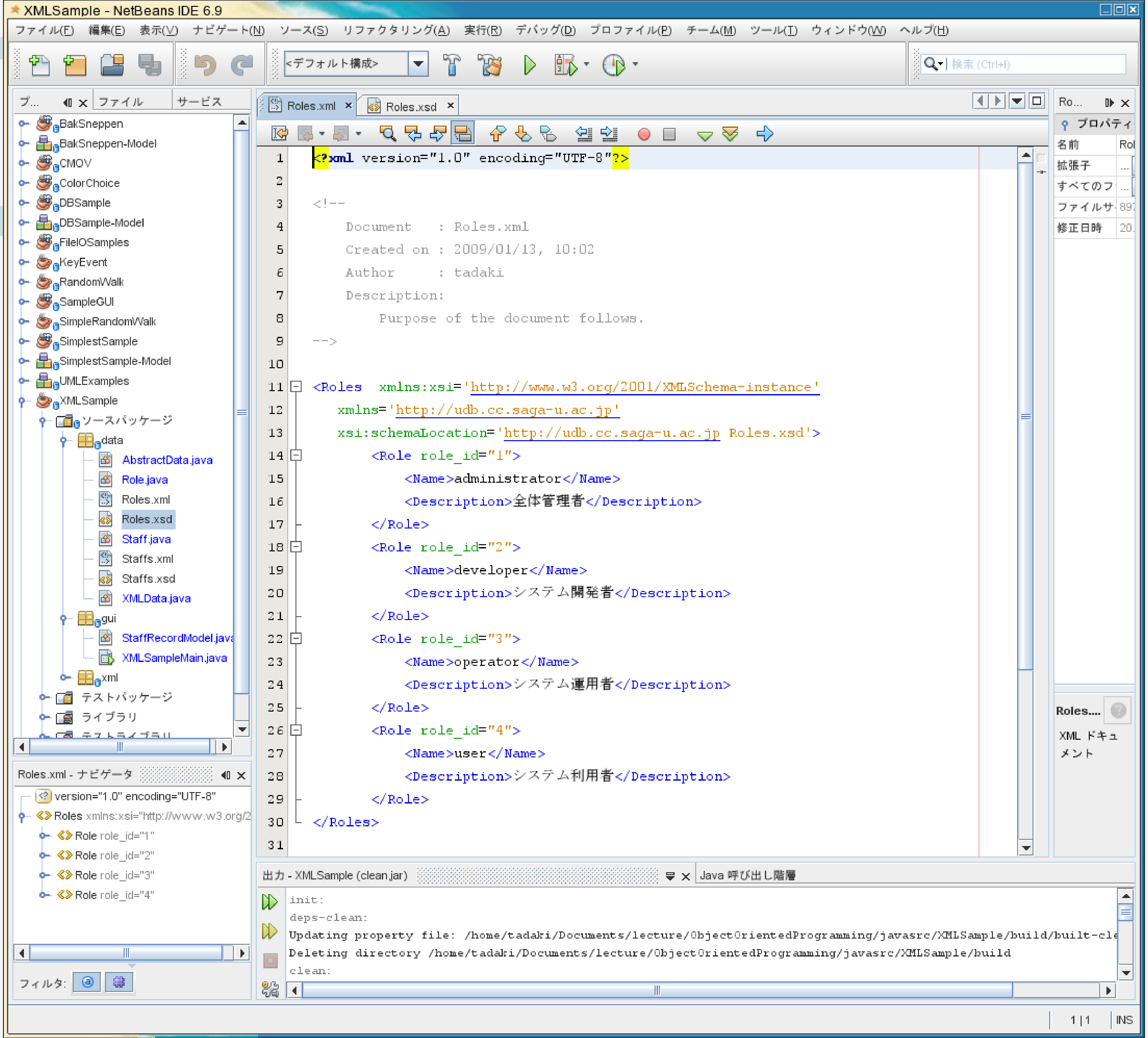
```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://udb.cc.saga-u.ac.jp"
    xmlns="http://udb.cc.saga-u.ac.jp"
    elementFormDefault="qualified">
    <xsd:element name="Staffs">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Staff" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Staff">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Name"/>
                <xsd:element ref="Description"/>
            </xsd:sequence>
            <xsd:attribute name="staff_id" type="xsd:int" use="required"/>
            <xsd:attribute name="valid" type="xsd:boolean" use="required"/>
            <xsd:attribute name="reg_date" type="xsd:dateTime" use="required"/>
            <xsd:attribute name="role" type="xsd:int" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Name" type="xsd:string"></xsd:element>
    <xsd:element name="Description" type="xsd:string"></xsd:element>
</xsd:schema>
```



```
init:
deps-clean:
Updating property file: /home/tadaki/Documents/lecture/ObjectOrientedProgramming/javasrc/XMLSample/build/built-clean.properties
Deleting directory /home/tadaki/Documents/lecture/ObjectOrientedProgramming/javasrc/XMLSample/build
clean:
init:
```



```
init:
deps-clean:
Updating property file: /home/tadaki/Documents/lecture/ObjectOrientedProgramming/javasrc/XMLSample/build/built-cle
Deleting directory /home/tadaki/Documents/lecture/ObjectOrientedProgramming/javasrc/XMLSample/build
clean:
```



```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : Roles.xml
    Created on : 2009/01/13, 10:02
    Author     : tadaki
    Description:
        Purpose of the document follows.
-->

<Roles  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns='http://udb.cc.saga-u.ac.jp'
   xsi:schemaLocation='http://udb.cc.saga-u.ac.jp Roles.xsd'>
       <Role role_id="1">
           <Name>administrator</Name>
           <Description>全体管理者</Description>
       </Role>
       <Role role_id="2">
           <Name>developer</Name>
           <Description>システム開発者</Description>
       </Role>
       <Role role_id="3">
           <Name>operator</Name>
           <Description>システム運用者</Description>
       </Role>
       <Role role_id="4">
           <Name>user</Name>
           <Description>システム利用者</Description>
       </Role>
</Roles>
```

```
init:
deps-clean:
Updating property file: /home/tadaki/Documents/lecture/ObjectOrientedProgramming/javasrc/XMLSample/build/built-cle
Deleting directory /home/tadaki/Documents/lecture/ObjectOrientedProgramming/javasrc/XMLSample/build
clean:
```

## XMLの構造を保持するクラス

- org.w3c.dom.Documentクラス
- SAX (Simple API for XML)
  - XMLを読む・書く
- DOM (Document Object Model)
  - XMLの構造を保持するモデル

## XMLデータ読み込みの流れ

- Fileからdom.Documentへ
- 要素名を指定して、該当するデータ一覧NodeList取得
    - 番号を指定して、一覧からNodeを取得
- getTextContentでNodeの文字内容を取得

## java.io.Fileから org.w3c.dom.Documentへ

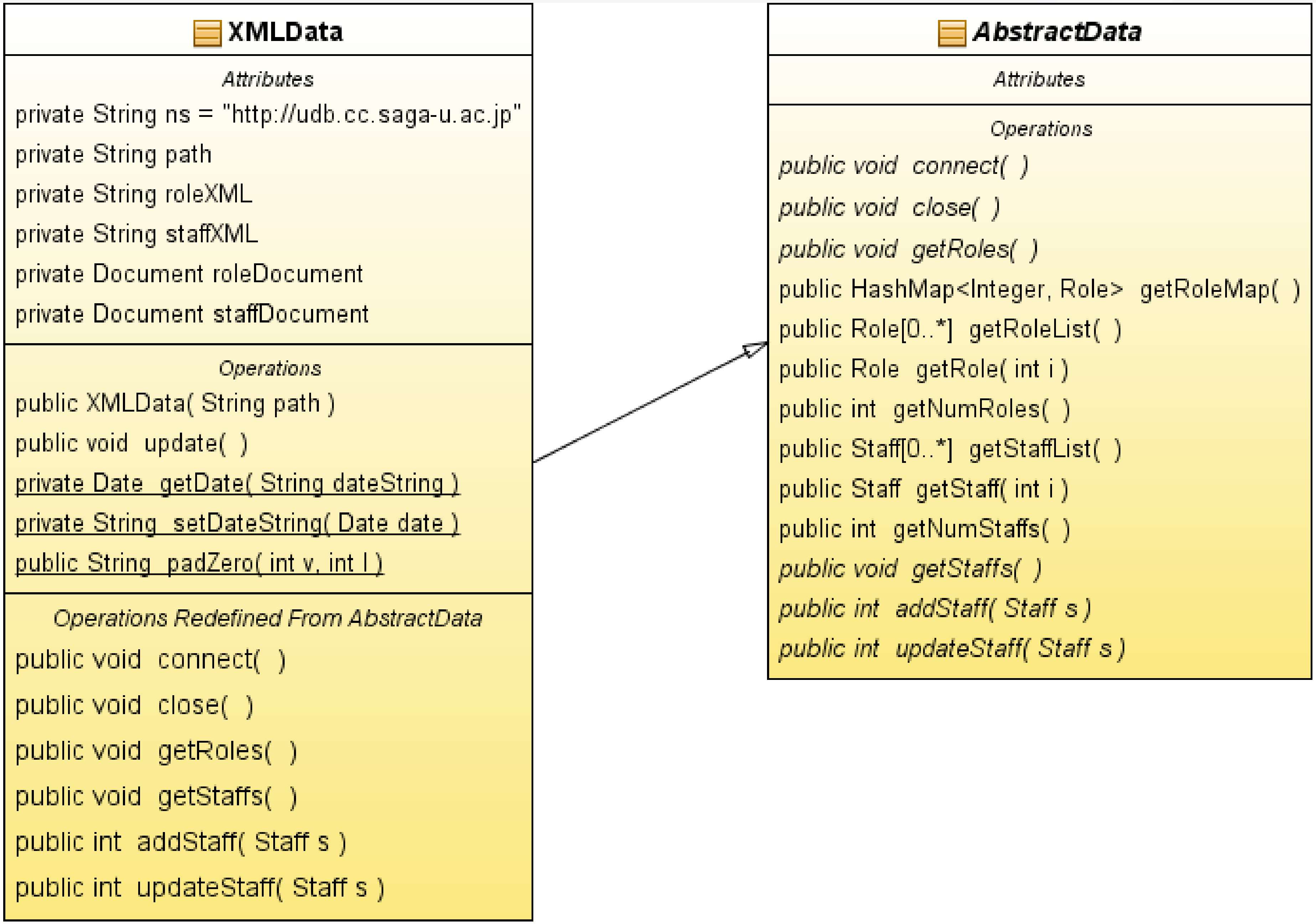
```
javax.xml.parsers.DocumentBuilderFactory   factory =
  javax.xml.parsers.DocumentBuilderFactory.newInstance();

javax.xml.parsers.DocumentBuilder builder = factory.newDocumentBuilder();

org.w3c.dom.Document document=builder.parse(file);
```

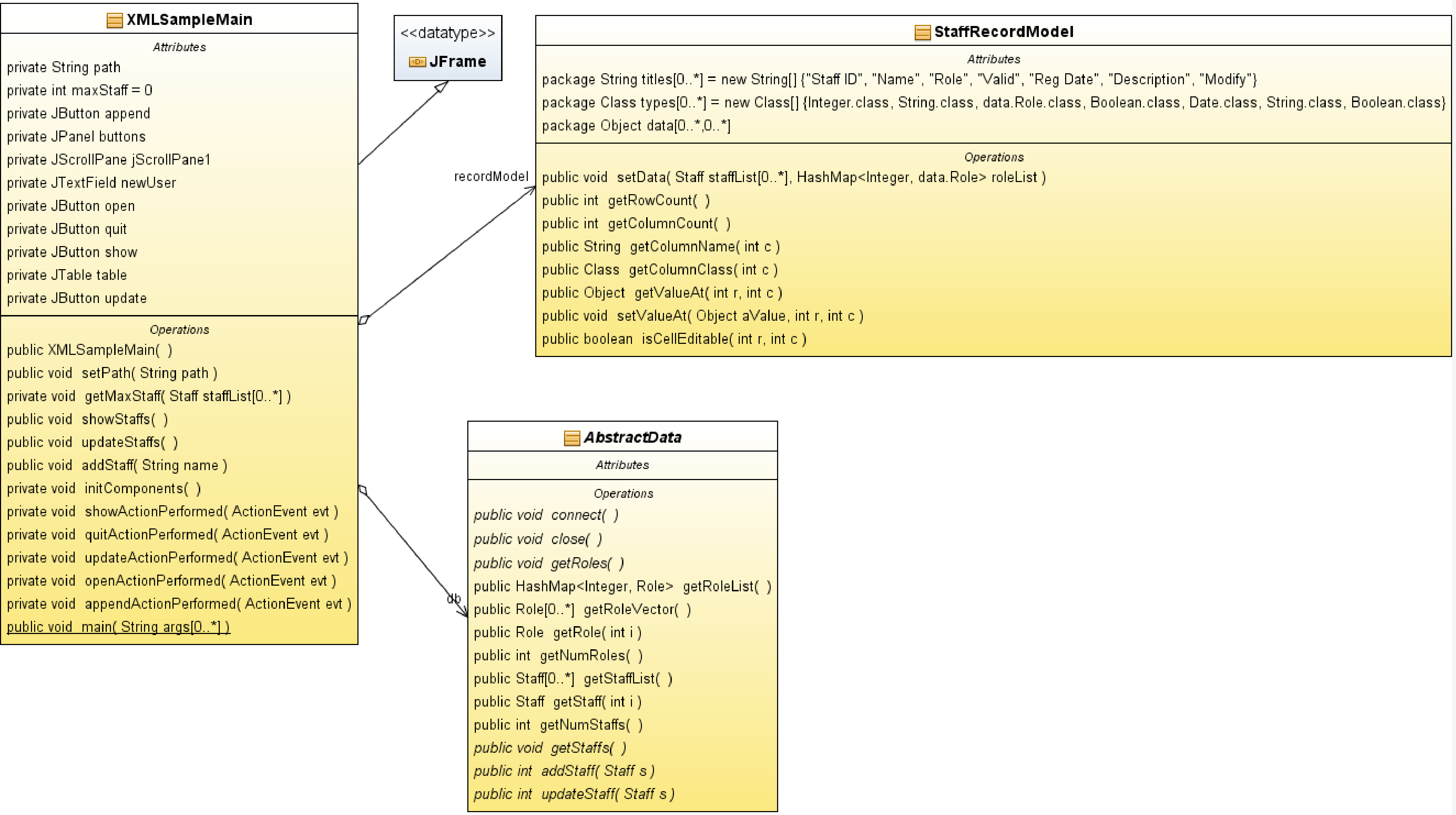
## データの取り出し

- 要素の指定
  - org.w3c.dom.NodeList nodeList = document.getElementsByTagNameNs(ns,要素名);
  - org.w3c.dom.Node node=nodeList.item(番号);
- 要素内の文字列取得
  - node.getTextContent();
- 要素の属性取得
  - node.getAttributes().getNamedItem(属性名).getTextContent()

## データの設定

- 要素内の文字指定
  - node.setTextContent(文字列);
- 要素の属性取得
  - node.getAttributes().getNamedItem(属性名).setTextContent(文字列)
- 子要素の追加
  - node.appendChild(子要素)


XMLData
Attributes
private String ns = "http://udb.cc.saga-u.ac.jp"
private String path
private String roleXML
private String staffXML
private Document roleDocument
private Document staffDocument
Operations
public XMLData( String path )
public void update( )
private Date getDate( String dateString )
private String setDateString( Date date )
public String padZero( int v, int l )
Operations Redefined From AbstractData
public void connect( )
public void close( )
public void getRoles( )
public void getStaffs( )
public int addStaff( Staff s )
public int updateStaff( Staff s )
AbstractData
Attributes
Operations
public void connect( )
public void close( )
public void getRoles( )
public HashMap<Integer, Role> getRoleMap( )
public Role[0..*] getRoleList( )
public Role getRole( int i )
public int getNumRoles( )
public Staff[0..*] getStaffList( )
public Staff getStaff( int i )
public int getNumStaffs( )
public void getStaffs( )
public int addStaff( Staff s )
public int updateStaff( Staff s )



```
XMLSampleMain
Attributes
private String path
private int maxStaff = 0
private JButton append
private JPanel buttons
private JScrollPane jScrollPane1
private JTextField newUser
private JButton open
private JButton quit
private JButton show
private JTable table
private JButton update
Operations
public XMLSampleMain( )
public void  setPath( String path )
private void  getMaxStaff( Staff staffList[0..*] )
public void  showStaffs( )
public void  updateStaffs( )
public void  addStaff( String name )
private void  initComponents( )
private void  showActionPerformed( ActionEvent evt )
private void  quitActionPerformed( ActionEvent evt )
private void  updateActionPerformed( ActionEvent evt )
private void  openActionPerformed( ActionEvent evt )
private void  appendActionPerformed( ActionEvent evt )
public void  main( String args[0..*] )
```

```
StaffRecordModel
Attributes
package String titles[0..*] = new String[] {"Staff ID", "Name", "Role", "Valid", "Reg Date", "Description", "Modify"}
package Class types[0..*] = new Class[] {Integer.class, String.class, data.Role.class, Boolean.class, Date.class, String.class, Boolean.class}
package Object data[0..*,0..*]
Operations
public void  setData( Staff staffList[0..*], HashMap<Integer, data.Role> roleList )
public int  getRowCount( )
public int  getColumnCount( )
public String  getColumnName( int c )
public Class  getColumnClass( int c )
public Object  getValueAt( int r, int c )
public void  setValueAt( Object aValue, int r, int c )
public boolean  isCellEditable( int r, int c )
```

```
AbstractData
Attributes
Operations
public void  connect( )
public void  close( )
public void  getRoles( )
public HashMap<Integer, Role>  getRoleList( )
public Role[0..*]  getRoleVector( )
public Role  getRole( int i )
public int  getNumRoles( )
public Staff[0..*]  getStaffList( )
public Staff  getStaff( int i )
public int  getNumStaffs( )
public void  getStaffs( )
public int  addStaff( Staff s )
public int  updateStaff( Staff s )
```

```
/**
 *
 * @author tadaki
 */

package data;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
abstract public class AbstractData {

    /**
     * Roleの一覧
     */
    protected HashMap<Integer, Role> roles;
    /**
     * Staffの一覧
     */
    protected HashMap<Integer, Staff> staffs;

    /**
     * データ源への接続
     * @throws java.lang.Exception
     */
    abstract public void connect() throws Exception;

    /**
     * データ源を閉じる
     * @throws java.lang.Exception
     */
    abstract public void close() throws Exception;

    /*
     * Role に関する検索
     */
    /**
     * Role一覧を取得
     * @throws java.lang.Exception
     */
    abstract public void getRoles() throws Exception;

    /**
```

```
     * Role一覧のマップのコピーを取得
     * @return マップのコピー
     */
    public HashMap<Integer, Role> getRoleMap() {
        if (roles == null) {
            return null;
        }
        HashMap<Integer, Role> m = new HashMap<Integer, Role>();
        for (Integer i : roles.keySet()) {
            m.put(i, roles.get(i));
        }
        return m;
    }

    /**
     * Role一覧のコピーをListとして返す
     * @return Role一覧のList
     */
    public List<Role> getRoleList() {
        if (roles == null) {
            return null;
        }
        List<Role> v = Collections.synchronizedList(new
 ArrayList<Role>());
        for (Integer i : roles.keySet()) {
            v.add(roles.get(i));
        }
        return v;
    }

    /**
     * Roleを取得する
     * @param i role_id
     * @return 取得したRole
     */
    public Role getRole(int i) {
        if (roles == null) {
            return null;
        }
        return roles.get(i);
    }

    /**
     * Roleの数を取得する
```

```
	 * @return Roleの数
	 */
	public int getNumRoles() {
		return roles.size();
	}
	/*
	 * Staff に関する検索
	 */

	/**
	 * Staff一覧をListとして取得する
	 * @return Staff一覧のList
	 */
	public List<Staff> getStaffList() {
		if (staffs == null) {
			return null;
		}
		List<Staff> s = Collections.synchronizedList(new
 ArrayList<Staff>());
		for (Integer i : staffs.keySet()) {
			s.add(staffs.get(i));
		}
		return s;
	}

	/**
	 * Staffを取得する
	 * @param i
	 * @return 取得したStaff
	 */
	public Staff getStaff(int i) {
		if (staffs == null) {
			return null;
		}
		return staffs.get(i);
	}

	/**
	 * Staffの数を得る
	 * @return Staffの数
	 */
	public int getNumStaffs() {
		return staffs.size();
	}
```

```
    /**
     * Staff一覧を取得
     * @throws java.lang.Exception
     */
    abstract public void getStaffs() throws Exception;

    /**
     * Staffを追加
     * @param s 追加するStaff
     * @throws java.lang.Exception
     */
    abstract public int addStaff(Staff s) throws Exception;

    /**
     * Staffの情報を更新する
     * @param s
     * @return 更新した数
     * @throws java.lang.Exception
     */
    abstract public int updateStaff(Staff s) throws Exception;
}
```

```
/**
 *
 * @author tadaki
 */
package data;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Date;
import java.util.Calendar;
import java.util.Collections;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;

public class XMLData extends AbstractData {

    private final String ns = "http://udb.cc.saga-u.ac.jp";//Namespace
    private String path;
    private String roleXML;
    private String staffXML;
    private Document roleDocument;
    private Document staffDocument;

    /**
     * コンストラクタ
     * @param path xmlファイルを含むディレクトリ名
     */
    public XMLData(String path) {
        this.path = path;
        roleXML = path + java.io.File.separator + "Roles.xml";
        staffXML = path + java.io.File.separator + "Staffs.xml";
    }

    public void connect() throws Exception {
        //XML ファイルからデータ読み込み
        roleDocument = new xml.XMLReader(roleXML).getDocument();
        staffDocument = new xml.XMLReader(staffXML).getDocument();
    }
```

```java
    public void close() throws Exception {
        update();
    }

    /**
     * ファイルへデータ書き出し
     * @throws java.lang.Exception
     */
    public void update() throws Exception {
        xml.XMLWriter writer = new xml.XMLWriter(staffXML);
        writer.setDocument(staffDocument);
        writer.putDomDocument();
    }

    public void getRoles() throws Exception {
        roles = new HashMap<Integer, Role>();
        //Roleタグの一覧取得
        NodeList list = roleDocument.getElementsByTagNameNS(ns, "Role");
        for (int i = 0; i < list.getLength(); i++) {
            //各Roleタグに対する処理
            Node node = list.item(i);
            String roleString =
                    node.getAttributes().getNamedItem("role_id").
                    getTextContent();
            int role_id = Integer.valueOf(roleString);
            Element e = (Element) node;
            Node nameNode = e.getElementsByTagNameNS(ns, "Name").item(0);
            String name = nameNode.getTextContent();
            Node descriptionNode =
                    e.getElementsByTagNameNS(ns, "Description").item(0);
            String description = descriptionNode.getTextContent();
            Role role = new Role(role_id, name, description);
            roles.put(role_id, role);
        }
    }

    @Override
    public void getStaffs() throws Exception {
        staffs = new HashMap<Integer, Staff>();
        //Staffタグの一覧取得
        NodeList list = staffDocument.getElementsByTagNameNS(ns,
"Staff");
        for (int i = 0; i < list.getLength(); i++) {
            //各Staffタグに対する処理
            Node node = list.item(i);
            //属性一覧
            HashMap<String, Node> attr = new HashMap<String, Node>();
            attr.put("staff_id", node.getAttributes().
                    getNamedItem("staff_id"));
            attr.put("valid",
node.getAttributes().getNamedItem("valid"));
            attr.put("reg_date", node.getAttributes().
                    getNamedItem("reg_date"));
            attr.put("role", node.getAttributes().getNamedItem("role"));

            int staff_id = Integer.valueOf(attr.get("staff_id").
                    getTextContent());
            boolean valid = Boolean.valueOf(attr.get("valid").
                    getTextContent());
            String dateString = attr.get("reg_date").getTextContent();
            Date reg_date = getDate(dateString);
            int role =
Integer.valueOf(attr.get("role").getTextContent());
            //子タグ処理
            Element e = (Element) node;
            Node nameNode =
                    e.getElementsByTagNameNS(ns, "Name").item(0);
            String name = nameNode.getTextContent();
            Node descNode =
                    e.getElementsByTagNameNS(ns, "Description").item(0);
            String description = descNode.getTextContent();
            Staff staff =
                    new Staff(staff_id, name, role, reg_date,
description);
            staff.setValid(valid);
            staffs.put(i, staff);
        }
    }

    public int addStaff(Staff s) throws Exception {
        //新しいタグの生成
        Element element = staffDocument.createElementNS(ns, "Staff");
        s.setReg_date(new Date());
        //属性の設定
        element.setAttribute("staff_id",
String.valueOf(s.getStaff_id()));
        element.setAttribute("valid", String.valueOf(s.isValid()));
        element.setAttribute("reg_date", setDateString(s.getReg_date()));
        element.setAttribute("role", String.valueOf(s.getRole()));
        //子タグの生成
        Element name = staffDocument.createElementNS(ns, "Name");
        name.setTextContent(s.getName());
        Element description =
                staffDocument.createElementNS(ns, "Description");
        description.setTextContent(s.getDescription());
        //属性及び子タグを追加
        element.appendChild(name);
        element.appendChild(description);
        //ドキュメントツリーに追加
        NodeList list = staffDocument.getElementsByTagNameNS(ns,
 "Staffs");
        list.item(0).appendChild(element);
        return 1;
    }

    public int updateStaff(Staff s) throws Exception {
        NodeList list = staffDocument.getElementsByTagNameNS(ns,
 "Staff");
        Node node = null;
        //対応するタグを検索
        for (int i = 0; i < list.getLength(); i++) {
            Node tmp = list.item(i);
            String str =
                    tmp.getAttributes().getNamedItem("staff_id").
                    getTextContent();
            int staff_id = Integer.valueOf(str);
            if (staff_id == s.getStaff_id()) {
                node = tmp;
            }
        }
        if (node == null) {
            return 0;
        }
        s.setReg_date(new Date());
        Element element = (Element) node;
        element.setAttribute("valid", String.valueOf(s.isValid()));
        element.setAttribute("reg_date", setDateString(s.getReg_date()));
        element.setAttribute("role", String.valueOf(s.getRole()));
        Node descNode =
                element.getElementsByTagNameNS(ns,
 "Description").item(0);
        descNode.setTextContent(s.getDescription());
        return 1;
    }

    /**
     * XML中の日付表現をjava.util.Dateへ変換
     * @param dateString XML中の日付表現文字列  2009-01-23T01:10:32
     * @return 変換されたDate型インスタンス
     */
    private static Date getDate(String dateString) {
        Calendar calendar = Calendar.getInstance();
        //数字を切り出す
        String patternString = "(¥¥d+)";
        Matcher m = Pattern.compile(patternString).matcher(dateString);
        List<Integer> ints =
                Collections.synchronizedList(new ArrayList<Integer>());
        while (m.find()) {
            ints.add(Integer.valueOf(m.group()));
        }
        int year = ints.get(0);
        int month = ints.get(1) - 1;
        int d = ints.get(2);
        int h = ints.get(3);
        int min = ints.get(4);
        int s = ints.get(5);
        calendar.set(year, month, d, h, min, s);
        return calendar.getTime();
    }

    /**
     * java.util.Date型からXML 向け日付表現
     * @param date Date型インスタンス
     * @return 変換された文字列
     */
    private static String setDateString(Date date) {
        StringBuilder buf = new StringBuilder();
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        int year = calendar.get(Calendar.YEAR);
        int month = calendar.get(Calendar.MONTH);
        int day = calendar.get(Calendar.DAY_OF_MONTH);
        int h = calendar.get(Calendar.HOUR_OF_DAY);
        int m = calendar.get(Calendar.MINUTE);
        int s = calendar.get(Calendar.SECOND);
        buf.append(year);
        buf.append("-").append(padZero(month + 1, 2));
        buf.append("-").append(padZero(day, 2));
        buf.append("T").append(padZero(h, 2));
        buf.append(":").append(padZero(m, 2));
        buf.append(":").append(padZero(s, 2));
        return buf.toString();
    }

    /**
     * 桁数を指定して、前に0 を補完
     * @param v 数値
     * @param l 桁数
     * @return  0を補完した文字列
     */
    static public String padZero(int v, int l) {
        String str = String.valueOf(v);
        int pl = l - str.length();
        if (pl <= 0) {
            return str;
        }
        StringBuilder b = new StringBuilder();
        for (int i = 0; i < pl; i++) {
            b.append("0");
        }
        b.append(str);
        return b.toString();
    }
}
```

DOMUtil.java

```java
/**
 *
 * @author tadaki
 */

package xml;

import java.io.*;
import org.w3c.dom.Document;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXParseException;
import org.xml.sax.SAXException;

public class DOMUtil {
        /**
     * Parse the XML file and create Document
     * @param fileName
     * @return Document
     */
    public static Document parse(String fileName)  throws Exception{
        return parse(new File(fileName));
    }

    /**
     * Parse the XML file and create Document
     * @param file
     * @return Document
     */
    public static Document parse(File file) throws Exception{
        Document document = null;
        // Initiate DocumentBuilderFactory
        DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();

        // To get a validating parser
        factory.setValidating(false);
        // To get one that understands namespaces
        factory.setNamespaceAware(true);

        try {
            // Get DocumentBuilder
            DocumentBuilder builder = factory.newDocumentBuilder();
            // Parse and load into memory the Document
            document = builder.parse(file);
            return document;

        } catch (SAXParseException spe) {
            // Error generated by the parser
            String st="¥n** Parsing error , line "
                    + spe.getLineNumber() + ", uri " + spe.getSystemId();
            System.err.println(st);
            System.err.println(" " + spe.getMessage());
            // Use the contained exception, if any
            Exception x = spe;
            if (spe.getException() != null) {
                x = spe.getException();
            }
            throw x;
        } catch (SAXException sxe) {
            // Error generated during parsing
            Exception x = sxe;
            if (sxe.getException() != null) {
                x = sxe.getException();
            }
            throw x;
        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            throw pce;
        } catch (IOException ioe) {
            // I/O error
            throw ioe;
        }
    }

}
```

XMLReader.java

```java
/**
 *
 * @author tadaki
 */
package xml;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

public class XMLReader {

    private String xmlFile = null;
    private Document document = null;

    public XMLReader(String xmlFile) throws Exception {
        this.xmlFile = xmlFile;
        document = getDomDocument(xmlFile);
    }

    public  Document getDocument() {
        return document;
    }

    public NodeList getNodeList(String name) {
        return document.getElementsByTagName(name);
    }

    public String getXmlFile() {
        return xmlFile;
    }

    public final Document getDomDocument(String xmlFile) throws
Exception {
        java.io.File file = null;
        file = new java.io.File(xmlFile);
        if (file == null) {
            System.exit(0);
        }
        return getDomDocument(file);
    }

    public Document getDomDocument(java.io.File file) throws Exception {
        document = DOMUtil.parse(file);
        return document;
```

```
      }
 }
```

XMLWriter.java

```java
/**
 *
 * @author tadaki
 */
package xml;

import java.io.IOException;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;

public class XMLWriter {

    private String xmlFile = null;
    protected org.w3c.dom.Document document = null;

    public XMLWriter(String xmlFile) {
        this.xmlFile = xmlFile;
    }

    public String getXmlFile() {
        return xmlFile;
    }

    public void putDomDocument()
            throws IOException,
            TransformerConfigurationException,
            TransformerException {
        java.io.File file = new java.io.File(xmlFile);
        if (file == null) {
            System.exit(0);
        }
        boolean newFile = true;
        if (!file.exists()) {
            newFile = file.createNewFile();
        }
        if (newFile) {
            putDomDocumentSub(file);
        }
    }

    public void setDocument(org.w3c.dom.Document document) {
        this.document = document;
    }
    protected void putDomDocumentSub(java.io.File file)
            throws TransformerConfigurationException,
            TransformerException {
        javax.xml.transform.Source source =
                new javax.xml.transform.dom.DOMSource(document);
        javax.xml.transform.Result result =
                new javax.xml.transform.stream.StreamResult(file);

        // Write the DOM document to the file
        // Get Transformer
        javax.xml.transform.Transformer xformer =
                javax.xml.transform.TransformerFactory.newInstance().
                newTransformer();
        xformer.transform(source, result);
    }
}
```