OdfSender.java

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.SwingUtilities;
import javax.swing.filechooser.*;
import javax.swing.tree.*;

public class OdfSender extends JFrame implements ActionListener{

      private JFileChooser fc;
      private JButton openButton, sendButton, clearButton;
      private String selectedDirectory;
      private JComboBox fileList;
      private JTextArea log;
      private JScrollPane logPane;

      public OdfSender() {
        //Window setup
        super("ODF Sender");
        setSize(550, 320);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

      //Entrance into application
      public static void main(String[] args){
            OdfSender selector = new OdfSender();
            selector.init();
            selector.setVisible(true);
      }

      public void init(){
            //file chooser allows directories only
            fc = new JFileChooser();
            fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            fileList = new JComboBox();
            Icon icon = UIManager.getIcon("FileView.directoryIcon"); //get
java directory icon
            //create buttons
            openButton = new JButton(icon);
            sendButton = new JButton("Send");
            sendButton.setVisible(false);
            clearButton = new JButton("Clear Log");
            //add action listeners
            openButton.addActionListener(this);
            sendButton.addActionListener(this);
            clearButton.addActionListener(this);
            //create log
            fileList.addItem("Select a Directory...");
            log = new JTextArea("Sending log: \n",12,42);
            log.setEditable(false);
            logPane = new JScrollPane(log);

            //create panels for layout
            JPanel mainPanel = new JPanel();
            mainPanel.setLayout(new BorderLayout());
            JPanel inputPanel = new JPanel(); //use FlowLayout
```

```
            JPanel logPanel = new JPanel();

            //add elements to panels
            mainPanel.add(inputPanel, BorderLayout.NORTH);
            mainPanel.add(logPanel, BorderLayout.CENTER);
          inputPanel.add(fileList);
            inputPanel.add(openButton);
            inputPanel.add(sendButton);
            logPanel.add(logPane);
            mainPanel.add(clearButton, BorderLayout.SOUTH);

            //add main panel to frame
            getContentPane().add(mainPanel);
      }

      public void actionPerformed(ActionEvent e) {
         //Handle open button action
        if (e.getSource() == openButton) {

                  int returnVal = fc.showOpenDialog(OdfSender.this);
            if (returnVal == JFileChooser.APPROVE_OPTION){
                        //get the directory name
                    File directory = fc.getSelectedFile();
                        selectedDirectory = directory.getAbsolutePath();
                        //clear the comboBox
                        fileList.removeAllItems();

                        //loop through directory and add each file to
comboBox
                        File[] listOfFiles = directory.listFiles();
                        for (int i = 0; i < listOfFiles.length; i++){
                              if (listOfFiles[i].isFile()){
                                    String s = new
String(listOfFiles[i].getName());
                                    fileList.addItem(s);
                              }
                        }
                        sendButton.setVisible(true);
                  }
            }
            //Handle send button action
            if (e.getSource() == sendButton) {
                  DefaultTreeModel tree;
                  //Get the filename
                  int i = fileList.getSelectedIndex();
                  String s = fileList.getItemAt(i).toString();
                  //add file name to log
                  log.append(s+" sent...\n");
                  //Create a SAXTreeViewer to parse the XML file
                  SAXTreeViewer viewer = new SAXTreeViewer();
                  try{
                        //create a tree from the XML file
                        tree = viewer.init(selectedDirectory+ "\\" +s);
                        JTree t = new JTree(tree);
                        JFrame f = new JFrame();
                        f.getContentPane().add(new JScrollPane(t),
                        BorderLayout.CENTER);
                        f.setVisible(true);
                        log.append("Success!\n");
                        OdfProcessor p = new OdfProcessor();
                        p.process(tree);
```

```
                    }
                    catch(Exception ex){
                          ex.printStackTrace();
                    }

                    //set the scroll pane to automatically scroll to the
bottom
                    JScrollBar vertical = logPane.getVerticalScrollBar();
                    vertical.setValue( vertical.getMaximum() );

                    //advance to next file in the list if not the last file
                    if(fileList.getSelectedIndex() !=
(fileList.getItemCount()-1)){
                          fileList.setSelectedIndex(i+1);
                    }
                    else{
                          //if it is the last file
                          log.append("END OF FILES\n");
                    }
              }

              if (e.getSource() == clearButton) {
                    //clear the log
                    log.setText("Sending log:\n");
              }
        }
}
```

SAXTreeViewer.java

```
/*--
     "This product includes software developed for the
      'Java and XML' book, by Brett McLaughlin (O'Reilly & Associates)."

        The class below is an adaptation of the SAXTreeViewer implemented
in the above book.
        I removed the JTree and JFrame elements so that the init method is
now used to return a DefaultMutableTree
        The nodes of this tree are of my own class- OdfElement
 */
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;

public class SAXTreeViewer extends JFrame {

    //Parser to use
    private String vendorParserClass =
"org.apache.xerces.parsers.SAXParser";
    //Tree model to use
    DefaultTreeModel defaultTreeModel;

    public SAXTreeViewer() {
    }

      //constructs the tree and returns it
    public DefaultTreeModel init(String xmlURI) throws IOException,
SAXException {
        DefaultMutableTreeNode base =
            new DefaultMutableTreeNode("XML Document: " +
                xmlURI);

        // Build the tree model
        defaultTreeModel = new DefaultTreeModel(base);


        // Construct the tree hierarchy
        buildTree(defaultTreeModel, base, xmlURI);

            return defaultTreeModel;
    }

    /**
     * <p>This handles building the Swing UI tree.</p>
     *
     * @param treeModel Swing component to build upon.
```

```
     * @param base tree node to build on.
     * @param xmlURI URI to build XML docum
       ent from.
     * @throws <code>IOException</code> - when reading the XML URI fails.
     * @throws <code>SAXException</code> - when errors in parsing occur.
     */
    public void buildTree(DefaultTreeModel treeModel,
                          DefaultMutableTreeNode base, String xmlURI)
        throws IOException, SAXException {

        // Create instances needed for parsing
        XMLReader reader =
            XMLReaderFactory.createXMLReader(vendorParserClass);
        ContentHandler jTreeContentHandler =
            new JTreeContentHandler(treeModel, base);
        ErrorHandler jTreeErrorHandler = new JTreeErrorHandler();

        // Register content handler
        reader.setContentHandler(jTreeContentHandler);

        // Register error handler
        reader.setErrorHandler(jTreeErrorHandler);

        // Parse
        InputSource inputSource =
            new InputSource(xmlURI);
        reader.parse(inputSource);
    }

}

/**
 * <b><code>JTreeContentHandler</code></b> implements the SAX
 *   <code>ContentHandler</code> interface and defines callback
 *   behavior for the SAX callbacks associated with an XML
 *   document's content, bulding up JTree nodes.
 */
class JTreeContentHandler implements ContentHandler {

    /** Hold onto the locator for location information */
    private Locator locator;

    /** Store URI to prefix mappings */
    private Map namespaceMappings;

    /** Tree Model to add nodes to */
    private DefaultTreeModel treeModel;

    /** Current node to add sub-nodes to */
    private DefaultMutableTreeNode current;

    /**
     * <p> Set up for working with the JTree. </p>
     *
     * @param treeModel tree to add nodes to.
     * @param base node to start adding sub-nodes to.
     */
    public JTreeContentHandler(DefaultTreeModel treeModel,
                               DefaultMutableTreeNode base) {
        this.treeModel = treeModel;
        this.current = base;
```

```
    this.namespaceMappings = new HashMap();
}

/**
 * <p>
 *  Provide reference to <code>Locator</code> which provides
 *    information about where in a document callbacks occur.
 * </p>
 *
 * @param locator <code>Locator</code> object tied to callback
 *        process
 */
public void setDocumentLocator(Locator locator) {
    // Save this for later use
    this.locator = locator;
}

/**
 * <p>
 *  This indicates the start of a Document parse-this precedes
 *    all callbacks in all SAX Handlers with the sole exception
 *    of <code>{@link #setDocumentLocator}</code>.
 * </p>
 *
 * @throws <code>SAXException</code> when things go wrong
 */
public void startDocument() throws SAXException {
    // No visual events occur here
}

/**
 * <p>
 *  This indicates the end of a Document parse-this occurs after
 *    all callbacks in all SAX Handlers.</code>.
 * </p>
 *
 * @throws <code>SAXException</code> when things go wrong
 */
public void endDocument() throws SAXException {
    // No visual events occur here
}

/**
 * <p>
 *   This indicates that a processing instruction (other than
 *     the XML declaration) has been encountered.
 * </p>
 *
 * @param target <code>String</code> target of PI
 * @param data <code>String</code containing all data sent to the PI.
 *               This typically looks like one or more attribute value
 *               pairs.
 * @throws <code>SAXException</code> when things go wrong
 */
public void processingInstruction(String target, String data)
    throws SAXException {

    DefaultMutableTreeNode pi =
        new DefaultMutableTreeNode("PI (target = '" + target +
                                   "', data = '" + data + "')");
    current.add(pi);
```

```
}

/**
 * <p>
 *   This indicates the beginning of an XML Namespace prefix
 *     mapping. Although this typically occurs within the root element
 *     of an XML document, it can occur at any point within the
 *     document. Note that a prefix mapping on an element triggers
 *     this callback <i>before</i> the callback for the actual element
 *     itself (<code>{@link #startElement}</code>) occurs.
 * </p>
 *
 * @param prefix <code>String</code> prefix used for the namespace
 *                being reported
 * @param uri <code>String</code> URI for the namespace
 *               being reported
 * @throws <code>SAXException</code> when things go wrong
 */
public void startPrefixMapping(String prefix, String uri) {
    // No visual events occur here.
    namespaceMappings.put(uri, prefix);
}

/**
 * <p>
 *   This indicates the end of a prefix mapping, when the namespace
 *     reported in a <code>{@link #startPrefixMapping}</code> callback
 *     is no longer available.
 * </p>
 *
 * @param prefix <code>String</code> of namespace being reported
 * @throws <code>SAXException</code> when things go wrong
 */
public void endPrefixMapping(String prefix) {
    // No visual events occur here.
    for (Iterator i = namespaceMappings.keySet().iterator();
         i.hasNext(); ) {

        String uri = (String)i.next();
        String thisPrefix = (String)namespaceMappings.get(uri);
        if (prefix.equals(thisPrefix)) {
            namespaceMappings.remove(uri);
            break;
        }
    }
}

/**
 * <p>
 *   This reports the occurrence of an actual element. It includes
 *     the element's attributes, with the exception of XML vocabulary
 *     specific attributes, such as
 *     <code>xmlns:[namespace prefix]</code> and
 *     <code>xsi:schemaLocation</code>.
 * </p>
 *
 * @param namespaceURI <code>String</code> namespace URI this element
 *               is associated with, or an empty <code>String</code>
 * @param localName <code>String</code> name of element (with no
 *               namespace prefix, if one is present)
 * @param qName <code>String</code> XML 1.0 version of element name:
```

```
     *                [namespace prefix]:[localName]
     * @param atts <code>Attributes</code> list for this element
     * @throws <code>SAXException</code> when things go wrong
     */

       //this method now creates the tree of OdfElements
       //we ignore namespaces and character data as they do not appear in
ODF messages
    public void startElement(String namespaceURI, String localName,
                             String qName, Attributes atts)
        throws SAXException {

        OdfElement o = new OdfElement(localName);

        // Process attributes
        for (int i=0; i<atts.getLength(); i++) {
            OdfAtt a = new OdfAtt(atts.getLocalName(i),atts.getValue(i));
            o.addAtt(a);
        }

            DefaultMutableTreeNode element = new DefaultMutableTreeNode(o);
        current.add(element);
        current = element;
    }

    /**
     * <p>
     *   Indicates the end of an element
     *     (<code>&lt;/[element name]&gt;</code>) is reached. Note that
     *     the parser does not distinguish between empty
     *     elements and non-empty elements, so this occurs uniformly.
     * </p>
     *
     * @param namespaceURI <code>String</code> URI of namespace this
     *                element is associated with
     * @param localName <code>String</code> name of element without prefix
     * @param qName <code>String</code> name of element in XML 1.0 form
     * @throws <code>SAXException</code> when things go wrong
     */
    public void endElement(String namespaceURI, String localName,
                           String qName)
        throws SAXException {

        // Walk back up the tree
        current = (DefaultMutableTreeNode)current.getParent();
    }

    /**
     * <p>
     *   This reports character data (within an element).
     * </p>
     *
     * @param ch <code>char[]</code> character array with character data
     * @param start <code>int</code> index in array where data starts.
     * @param length <code>int</code> index in array where data ends.
     * @throws <code>SAXException</code> when things go wrong
     */
    public void characters(char[] ch, int start, int length)
        throws SAXException {
    }
```

```
    /**
     * <p>
     * This reports whitespace that can be ignored in the
     * originating document. This is typically invoked only when
     * validation is ocurring in the parsing process.
     * </p>
     *
     * @param ch <code>char[]</code> character array with character data
     * @param start <code>int</code> index in array where data starts.
     * @param end <code>int</code> index in array where data ends.
     * @throws <code>SAXException</code> when things go wrong
     */
    public void ignorableWhitespace(char[] ch, int start, int length)
        throws SAXException {

        // This is ignorable, so don't display it
    }

    /**
     * <p>
     *   This reports an entity that is skipped by the parser. This
     *     should only occur for non-validating parsers, and then is still
     *     implementation-dependent behavior.
     * </p>
     *
     * @param name <code>String</code> name of entity being skipped
     * @throws <code>SAXException</code> when things go wrong
     */
    public void skippedEntity(String name) throws SAXException {
        DefaultMutableTreeNode skipped =
            new DefaultMutableTreeNode("Skipped Entity: '" + name + "'");
        current.add(skipped);
    }
}

/**
 * <b><code>JTreeErrorHandler</code></b> implements the SAX
 *   <code>ErrorHandler</code> interface and defines callback
 *   behavior for the SAX callbacks associated with an XML
 *   document's warnings and errors.
 */
class JTreeErrorHandler implements ErrorHandler {

    /**
     * <p>
     * This will report a warning that has occurred; this indicates
     *   that while no XML rules were "broken", something appears
     *   to be incorrect or missing.
     * </p>
     *
     * @param exception <code>SAXParseException</code> that occurred.
     * @throws <code>SAXException</code> when things go wrong
     */
    public void warning(SAXParseException exception)
        throws SAXException {

        System.out.println("**Parsing Warning**\n" +
                           "  Line:    " +
                           exception.getLineNumber() + "\n" +
                           "  URI:     " +
                           exception.getSystemId() + "\n" +
```

```
                            "  Message: " +
                               exception.getMessage());
            throw new SAXException("Warning encountered");
        }

        /**
         * <p>
         * This will report an error that has occurred; this indicates
         *   that a rule was broken, typically in validation, but that
         *   parsing can reasonably continue.
         * </p>
         *
         * @param exception <code>SAXParseException</code> that occurred.
         * @throws <code>SAXException</code> when things go wrong
         */
        public void error(SAXParseException exception)
            throws SAXException {

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

        /**
         * <p>
         * This will report a fatal error that has occurred; this indicates
         *   that a rule has been broken that makes continued parsing either
         *   impossible or an almost certain waste of time.
         * </p>
         *
         * @param exception <code>SAXParseException</code> that occurred.
         * @throws <code>SAXException</code> when things go wrong
         */
        public void fatalError(SAXParseException exception)
            throws SAXException {

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

OdfElement.java

```
import java.util.Vector;

//odf element stores the element name and a vector containing its
attributes

public class OdfElement{

	private String ElementName;
	private Vector<OdfAtt> AttributeList;

	public OdfElement(String ElementName){
		this.ElementName = ElementName;
		AttributeList = new Vector<OdfAtt>();
	}

	public void addAtt(OdfAtt attribute){
		AttributeList.add(attribute);
	}

	public String toString(){
		String attString = new String();
		for (int i = 0; i <AttributeList.size();i++){
			String a = new String((AttributeList.get(i).toString()));
			attString = attString.concat(a);
		}
		String s = new String(ElementName+"- "+attString);
		return s;
	}

	public String getElementName(){
		return ElementName;
	}

	public String getAttValue(String searchTerm){
		//search through the vector
		String result = " ";
		for (int i = 0; i <AttributeList.size();i++){
			OdfAtt a = AttributeList.get(i);
			String attName = a.getAttName();
			//if the search term matches the attribute name, return
the attributes value
			if (attName.equals(searchTerm)){
				result = a.getAttValue();
				break;
			}
		}
		return result;
	}
}
```

OdfAtt.java

```
//each attribute is a pair of strings, the attribute name and its value
public class OdfAtt{

	private String AttName;
	private String AttValue;

	public OdfAtt(String AttName, String AttValue){
		this.AttName = AttName;
		this.AttValue = AttValue;
	}

	public String toString(){
		String s = (AttName + ": "+AttValue+", ");
		return s;
	}

	public String getAttName(){
		return AttName;
	}

	public String getAttValue(){
		return AttValue;
	}
}
```

OdfProcessor.java

```
import java.sql.*;
import javax.swing.tree.*;

public class OdfProcessor{

	private Connection con;
	private Statement st;
	private String DocType;
	private String DocCode;
	private DefaultMutableTreeNode current;
	private String sql;

	public OdfProcessor(){
		init();
	}

	public void init(){
		try {
			Class.forName("com.mysql.jdbc.Driver");
			con =
DriverManager.getConnection("jdbc:mysql://localhost/odf", "root",
"H3liC0pter");
			st = con.createStatement();
			/*
			String sql = ("INSERT INTO ATHLETES
VALUES('1234567','Smith','Bob','M','1990-09-18','ESP')");
			st.executeUpdate(sql);
			System.out.println("Connected to the database");
			con.close();
			System.out.println("Disconnected from database");
			*/
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}

	public void process(DefaultTreeModel tree){
		//get the root
		current = (DefaultMutableTreeNode)tree.getRoot();
		//go to child and get document type and event code
		current = (DefaultMutableTreeNode)current.getFirstChild();
		OdfElement o =(OdfElement) current.getUserObject();
		DocType = o.getAttValue("DocumentType");
		DocCode = o.getAttValue("DocumentCode");
		//call processing instructions depending on Document Type
		if (DocType.equals("DT_PARTIC") ||
DocType.equals("DT_PARTIC_UPDATE")){
			processDT_PARTIC(current);
		}
		else if (DocType.equals("DT_PARTIC_TEAMS") ||
DocType.equals("DT_PARTIC_TEAMS_UPDATE")){
			processDT_PARTIC_TEAMS(current);
		}
		else if (DocType.equals("DT_START_LIST")){
			processDT_START_LIST(current);
		}
	}
```

```
	public void processDT_PARTIC(DefaultMutableTreeNode current2){
		//go to child node (OdfBody)
		current2 = (DefaultMutableTreeNode)current2.getFirstChild();
		//go to child node (Participant)
		current2 = (DefaultMutableTreeNode)current2.getFirstChild();
		String athleteFunction;
		String athleteAccred;
		//loop through all participants
		//if the particpants is an accredited athlete, insert data
		for(int i = 0; i < current2.getSiblingCount();i++){
			OdfElement o =(OdfElement) current2.getUserObject();
			athleteFunction = o.getAttValue("MainFunctionId");
			athleteAccred = o.getAttValue("Status");
			if(athleteFunction.equals("AA01") &&
athleteAccred.equals("ACCRED") ){
				String athleteCode = o.getAttValue("Code");
				String athleteFName = o.getAttValue("FamilyName");
				String athleteGName = o.getAttValue("GivenName");
				String athleteGender = o.getAttValue("Gender");
				String athleteDOB = o.getAttValue("BirthDate");
				athleteDOB = new StringBuffer(athleteDOB).insert(4,
"-").toString(); //convert DOB to correct format
				athleteDOB = new StringBuffer(athleteDOB).insert(7,
"-").toString();
				String athleteOrg = o.getAttValue("Organisation");

				//SQL insert
				sql = ("INSERT INTO ATHLETES VALUES('"
						+athleteCode+"','"
						+athleteFName+"','"
						+athleteGName+"','"
						+athleteGender+"','"
						+athleteDOB+"','"
						+athleteOrg+"')");
				try{
					st.executeUpdate(sql);
				}
				catch (Exception e){
					e.printStackTrace();
				}
			}
			//move to next sibling node, if not the last node
			if( i != current2.getSiblingCount()-1){
				current2 =
(DefaultMutableTreeNode)current2.getNextSibling();
			}
		}
	}

	public void processDT_PARTIC_TEAMS(DefaultMutableTreeNode current2){
		//go to child node (OdfBody)
		current2 = (DefaultMutableTreeNode)current2.getFirstChild();
		//go to child node (Team)
		current2 = (DefaultMutableTreeNode)current2.getFirstChild();
		//loop through teams
		for(int i = 0; i < current2.getSiblingCount();i++){
			OdfElement o =(OdfElement) current2.getUserObject();
			String teamCode = o.getAttValue("Code");
			String teamOrg = o.getAttValue("Organisation");
			String teamName = o.getAttValue("Name");
			//go to childe node (Composition)
```

```
                  current2 =
(DefaultMutableTreeNode)current2.getFirstChild();
                  //go to childe node (Athlete)
                  current2 =
(DefaultMutableTreeNode)current2.getFirstChild();
                  o =(OdfElement) current2.getUserObject();
                  String teamAthlete1 = o.getAttValue("Code");
                  current2 =
(DefaultMutableTreeNode)current2.getNextSibling();
                  o =(OdfElement) current2.getUserObject();
                  String teamAthlete2 = o.getAttValue("Code");
                  //back to Team node
                  current2 = (DefaultMutableTreeNode)current2.getParent();
                  current2 = (DefaultMutableTreeNode)current2.getParent();

                  //SQL insert
                        sql = ("INSERT INTO TEAMS VALUES('"
                                    +teamCode+"','"
                                    +teamOrg+"','"
                                    +teamName+"','"
                                    +teamAthlete1+"','"
                                    +teamAthlete2+"')");
                        try{
                              st.executeUpdate(sql);
                        }
                        catch (Exception e){
                              e.printStackTrace();
                        }

                  //move to next sibling node, if not the last node
                  if( i != current2.getSiblingCount()-1){
                        current2 =
(DefaultMutableTreeNode)current2.getNextSibling();
                  }
            }
      }

      public void processDT_START_LIST(DefaultMutableTreeNode current2){
            //go to child node (OdfBody)
            current2 = (DefaultMutableTreeNode)current2.getFirstChild();
            //go to child node (UnitInfos-same level as start)
            current2 = (DefaultMutableTreeNode)current2.getFirstChild();
            //loop through nodes
            for(int i = 0; i < current2.getSiblingCount();i++){
                  OdfElement o =(OdfElement) current2.getUserObject();
                  //if the element is "start"
                  if(o.getElementName().equals("Start")){
                        //get start order
                        String startOrder = o.getAttValue("StartOrder");
                        //go to next child and get competitor code
                        current2 =
(DefaultMutableTreeNode)current2.getFirstChild();
                        o =(OdfElement) current2.getUserObject();
                        String competitorCode = o.getAttValue("Code");

                        //insert start list into results
                        sql = ("INSERT INTO RESULTS VALUES('"
                                    +DocCode+"','"
                                    +competitorCode+"','"
                                    +startOrder+"')");
                        try{
```

```
                        st.executeUpdate(sql);
                  }
                  catch (Exception e){
                        e.printStackTrace();
                  }


                  //DIVES
                  String diveCode1 = "none";
                  String diveDiff1 = "none";
                  String diveDesc1 = "none";
                  String diveCode2 = "none";
                  String diveDiff2 = "none";
                  String diveDesc2 = "none";
                  String diveCode3 = "none";
                  String diveDiff3 = "none";
                  String diveDesc3 = "none";
                  String diveCode4 = "none";
                  String diveDiff4 = "none";
                  String diveDesc4 = "none";
                  String diveCode5 = "none";
                  String diveDiff5 = "none";
                  String diveDesc5 = "none";
                  String diveCode6 = "none";
                  String diveDiff6 = "none";
                  String diveDesc6 = "none";
                  current2 =
(DefaultMutableTreeNode)current2.getFirstChild();
                  for(int j = 0; j < current2.getSiblingCount();j++){


                        o =(OdfElement) current2.getUserObject();

      if(o.getElementName().equals("EventUnitEntry")){

      if(o.getAttValue("Code").equals("DV_NUMBER") &&
(o.getAttValue("Pos").equals("1"))){
                                    diveCode1 =
o.getAttValue("Value");
                              }
                              else
if(o.getAttValue("Code").equals("DV_DD") &&
(o.getAttValue("Pos").equals("1"))){
                                    diveDiff1 =
o.getAttValue("Value");
                              }
                              else
if(o.getAttValue("Code").equals("DV_DESCRIPTION") &&
(o.getAttValue("Pos").equals("1"))){
                                    diveDesc1 =
o.getAttValue("Value");
                              }
                              else
if(o.getAttValue("Code").equals("DV_NUMBER") &&
(o.getAttValue("Pos").equals("2"))){
                                    diveCode2 =
o.getAttValue("Value");
                              }
                              else
if(o.getAttValue("Code").equals("DV_DD") &&
(o.getAttValue("Pos").equals("2"))){
```

```
                                        diveDiff2 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DESCRIPTION") &&
(o.getAttValue("Pos").equals("2"))){
                                        diveDesc2 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_NUMBER") &&
(o.getAttValue("Pos").equals("3"))){
                                        diveCode3 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DD") &&
(o.getAttValue("Pos").equals("3"))){
                                        diveDiff3 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DESCRIPTION") &&
(o.getAttValue("Pos").equals("3"))){
                                        diveDesc3 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_NUMBER") &&
(o.getAttValue("Pos").equals("4"))){
                                        diveCode4 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DD") &&
(o.getAttValue("Pos").equals("4"))){
                                        diveDiff4 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DESCRIPTION") &&
(o.getAttValue("Pos").equals("4"))){
                                        diveDesc4 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_NUMBER") &&
(o.getAttValue("Pos").equals("5"))){
                                        diveCode5 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DD") &&
(o.getAttValue("Pos").equals("5"))){
                                        diveDiff5 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DESCRIPTION") &&
(o.getAttValue("Pos").equals("5"))){
```

```
                                        diveDesc5 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_NUMBER") &&
(o.getAttValue("Pos").equals("6"))){
                                        diveCode6 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DD") &&
(o.getAttValue("Pos").equals("6"))){
                                        diveDiff6 =
o.getAttValue("Value");
                                    }
                                    else
if(o.getAttValue("Code").equals("DV_DESCRIPTION") &&
(o.getAttValue("Pos").equals("6"))){
                                        diveDesc6 =
o.getAttValue("Value");
                                    }

                                }
                                //go to next sibling
                                if( j != current2.getSiblingCount()-1){
                                    current2 =
(DefaultMutableTreeNode)current2.getNextSibling();
                                }

                            }
                            //insert dives
                            //if dive 6 is none, then insert 5 dives, else
insert 6
                            if (diveCode6.equals("none")){
                                sql = ("INSERT INTO DIVES
(competitor,dive_number,dive_code,description,difficulty) VALUES ('"
                                    +competitorCode+"','1','"
                                    +diveCode1+"','"
                                    +diveDesc1+"','"
                                    +diveDiff1+"'),('"
                                    +competitorCode+"','2','"
                                    +diveCode2+"','"
                                    +diveDesc2+"','"
                                    +diveDiff2+"'),('"
                                    +competitorCode+"','3','"
                                    +diveCode3+"','"
                                    +diveDesc3+"','"
                                    +diveDiff3+"'),('"
                                    +competitorCode+"','4','"
                                    +diveCode4+"','"
                                    +diveDesc4+"','"
                                    +diveDiff4+"'),('"
                                    +competitorCode+"','5','"
                                    +diveCode5+"','"
                                    +diveDesc5+"','"
                                    +diveDiff5+"')");
                            }
                            else{
                                sql = ("INSERT INTO DIVES
(competitor,dive_number,dive_code,description,difficulty) VALUES ('"
                                    +competitorCode+"','1','"
```

```
                                    +diveCode1+"','"
                                    +diveDesc1+"','"
                                    +diveDiff1+"'),('"
                                    +competitorCode+"','2','"
                                    +diveCode2+"','"
                                    +diveDesc2+"','"
                                    +diveDiff2+"'),('"
                                    +competitorCode+"','3','"
                                    +diveCode3+"','"
                                    +diveDesc3+"','"
                                    +diveDiff3+"'),('"
                                    +competitorCode+"','4','"
                                    +diveCode4+"','"
                                    +diveDesc4+"','"
                                    +diveDiff4+"'),('"
                                    +competitorCode+"','5','"
                                    +diveCode5+"','"
                                    +diveDesc5+"','"
                                    +diveDiff5+"'),('"
                                    +competitorCode+"','6','"
                                    +diveCode6+"','"
                                    +diveDesc6+"','"
                                    +diveDiff6+"')");

                        }
                              try{
                                    st.executeUpdate(sql);
                              }
                              catch (Exception e){
                                    e.printStackTrace();
                              }

                        current2 =
(DefaultMutableTreeNode)current2.getParent();
                        //END OF DIVE PROCESSING




                        //back to "start" nodes
                        current2 =
(DefaultMutableTreeNode)current2.getParent();
                  }
                  if( i != current2.getSiblingCount()-1){
                        current2 =
(DefaultMutableTreeNode)current2.getNextSibling();
                  }
            }
      }
}
```