ListHierarchyPathToRoot Java

Java Code

```
/*
 * Copyright: (c) 2004-2009 Mayo Foundation for Medical Education and
 * Research (MFMER). All rights reserved. MAYO, MAYO CLINIC, and the
 * triple-shield Mayo logo are trademarks and service marks of MFMER.
 *
 * Except as contained in the copyright notice above, or as used to identify
 * MFMER as the author of this software, the trade names, trademarks, service
 * marks, or product names of the copyright holder shall not be used in
 * advertising, promotion or otherwise in connection with this software without
 * prior written authorization of the copyright holder.
 *
 * Licensed under the Eclipse Public License, Version 1.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *                 http://www.eclipse.org/legal/epl-v10.html
 *
 */
package org.LexGrid.LexBIG.example;

import org.LexGrid.LexBIG.DataModel.Collections.AssociatedConceptList;
import org.LexGrid.LexBIG.DataModel.Collections.AssociationList;
import org.LexGrid.LexBIG.DataModel.Core.AssociatedConcept;
import org.LexGrid.LexBIG.DataModel.Core.Association;
import org.LexGrid.LexBIG.DataModel.Core.CodingSchemeSummary;
import org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag;
import org.LexGrid.LexBIG.Exceptions.LBException;
import org.LexGrid.LexBIG.Extensions.Generic.LexBIGServiceConvenienceMethods;
import org.LexGrid.LexBIG.Impl.LexBIGServiceImpl;
import org.LexGrid.LexBIG.LexBIGService.LexBIGService;

/**
 * Example showing how to determine and display paths from a given concept back
 * to defined root nodes through any hierarchies registered for the coding
 * scheme.
 *
 * This program accepts one parameter (required), indicating the code to
 * evaluate.
 *
 * BACKGROUND: From a database perspective, LexBIG stores relationships
 * internally in a forward direction, source to target. Due to differences in
 * source formats, however, a wide variety of associations may be used ('PAR',
 * 'CHD', 'isa', 'hasSubtype', etc). In addition, the direction of navigation
 * may vary ('isa' expands in a reverse direction whereas 'hasSubtype' expands
 * in a forward direction.
 *
 * The intent of the getHierarchy* methods on the
 * LexBIGServiceConvenienceMethods interface is to simplify the process of
 * hierarchy discovery and navigation. These methods significantly reduce the
 * need to understand conventions for root nodes, associations, and direction of
 * navigation for a specific source format.
 *
 */
public class ListHierarchyPathToRoot {

    public ListHierarchyPathToRoot() {
        super();
    }

    /**
     * Entry point for processing.
     *
     * @param args
     */
```

```
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Example: ListHierarchyPathToRoot \"C0000\"");
            return;
        }
        ;

        try {
            String code = args[0];
            new ListHierarchyPathToRoot().run(code);
        } catch (Exception e) {
            Util.displayAndLogError("REQUEST FAILED !!!", e);
        }
    }

    public void run(String code) throws LBException {
        CodingSchemeSummary css = Util.promptForCodeSystem();
        long ms = System.currentTimeMillis();
        try {
            if (css != null) {
                LexBIGService lbSvc = LexBIGServiceImpl.defaultInstance();
                LexBIGServiceConvenienceMethods lbscm = (LexBIGServiceConvenienceMethods) lbSvc
                        .getGenericExtension("LexBIGServiceConvenienceMethods");

                // Fetch the description for the specified code.
                // Not required to find path to root, just nice to display.
                String scheme = css.getCodingSchemeURI();
                CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
                csvt.setVersion(css.getRepresentsVersion());
                String desc = null;
                try {
                    desc = lbscm.createCodeNodeSet(new String[] { code }, scheme, csvt).resolveToList(null,
null, null,
                            1).getResolvedConceptReference(0).getEntityDescription().getContent();
                } catch (Exception e) {
                    desc = "<not found>";
                }
                Util.displayMessage("============================================================");
                Util.displayMessage("Focus code: " + code + ":" + desc);
                Util.displayMessage("============================================================");

                // Iterate through all hierarchies ...
                String[] hierarchyIDs = lbscm.getHierarchyIDs(scheme, csvt);
                for (int i = 0; i < hierarchyIDs.length; i++) {
                    String hierarchyID = hierarchyIDs[i];
                    Util.displayMessage("------------------------------------------------------------");
                    Util.displayMessage("Hierarchy ID: " + hierarchyID);
                    Util.displayMessage("------------------------------------------------------------");
                    AssociationList associations = lbscm.getHierarchyPathToRoot(scheme, csvt, hierarchyID,
code, false,
                            LexBIGServiceConvenienceMethods.HierarchyPathResolveOption.ALL, null);
                    for (int j = 0; j < associations.getAssociationCount(); j++) {
                        Association association = associations.getAassociation(j);
                        printChain(association, 0);
                    }
                    Util.displayMessage("");
                }
            }
        } finally {
            System.out.println("Run time (ms): " + (System.currentTimeMillis() - ms));
        }
    }

    /**
     * Displays the given concept chain, taking into account any branches that
     * might be imbedded.
     *
     * @param assoc
     * @param depth
     * @throws LBException
     */
```

```
    protected void printChain(Association assoc, int depth) throws LBException {
        StringBuffer indent = new StringBuffer();
        for (int i = 0; i <= depth; i++)
            indent.append("    ");

        AssociatedConceptList concepts = assoc.getAssociatedConcepts();
        for (int i = 0; i < concepts.getAssociatedConceptCount(); i++) {
            // Print focus of this branch ...
            AssociatedConcept concept = concepts.getAssociatedConcept(i);
            Util.displayMessage(new StringBuffer(indent).append(assoc.getAssociationName()).append("->").append(
                    concept.getConceptCode()).append(':').append(
                    concept.getEntityDescription() == null ? "NO DESCRIPTION" : concept.getEntityDescription()
                            .getContent()).toString());

            // Find and recurse printing for next batch ...
            AssociationList nextLevel = concept.getSourceOf();
            if (nextLevel != null && nextLevel.getAssociationCount() != 0)
                for (int j = 0; j < nextLevel.getAssociationCount(); j++)
                    printChain(nextLevel.getAssociation(j), depth + 1);
        }
    }
}
```