FindUMLSContextsForCUI 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.AssociationList;
import org.LexGrid.LexBIG.DataModel.Collections.ResolvedConceptReferenceList;
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.DataModel.Core.ResolvedConceptReference;
import org.LexGrid.LexBIG.Exceptions.LBException;
import org.LexGrid.LexBIG.Impl.LexBIGServiceImpl;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeGraph;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeSet;
import org.LexGrid.LexBIG.LexBIGService.LexBIGService;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeSet.PropertyType;
import org.LexGrid.LexBIG.Utility.ConvenienceMethods;
import org.LexGrid.commonTypes.PropertyQualifier;
import org.LexGrid.commonTypes.Text;
import org.LexGrid.concepts.Concept;
import org.LexGrid.concepts.Entity;
import org.LexGrid.concepts.Presentation;

/**
 * Example showing any source-asserted hierarchies (based on import of MRHIER
 * HCD) for a CUI. The program takes a single argument (the UMLS CUI), prompts
 * for the code system to query in the LexGrid repository, and displays the
 * available hierarchical relationships.
 */
public class FindUMLSContextsForCUI {
    static String[] hierAssocNames_ = new String[] { "PAR", "CHD", "hasSubtype", "isa", "branch_of", "part_of",
            "tributary_of" };

    public FindUMLSContextsForCUI() {
        super();
    }

    /**
     * Entry point for processing.
     *
     * @param args
     */
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Example: FindUMLSContextsForCUI \"C0030920\"");
            return;
        }
```

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

/**
 * Process the provided code.
 *
 * @param code
 * @throws LBException
 */
public void run(String cui) throws LBException {
    CodingSchemeSummary css = Util.promptForCodeSystem();
    if (css != null) {
        LexBIGService lbSvc = LexBIGServiceImpl.defaultInstance();
        String scheme = css.getCodingSchemeURI();
        CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
        csvt.setVersion(css.getRepresentsVersion());

        ResolvedConceptReference ref = printCode(cui, lbSvc, scheme, csvt);
        if (ref != null)
            printContext(ref, lbSvc, scheme, csvt);
    }
}

/**
 * Display name and description information for the given cui.
 *
 * @param code
 * @param lbSvc
 * @param scheme
 * @param csvt
 * @return A fully resolved concept reference for the code.
 * @throws LBException
 */
protected ResolvedConceptReference printCode(String cui, LexBIGService lbSvc, String scheme,
        CodingSchemeVersionOrTag csvt) throws LBException {
    // Locate the matching concept ...
    CodedNodeSet cns = lbSvc.getCodingSchemeConcepts(scheme, csvt);
    cns.restrictToMatchingProperties(ConvenienceMethods.createLocalNameList("UMLS_CUI"),
            new PropertyType[] { PropertyType.GENERIC }, cui, "exactMatch", null);
    ResolvedConceptReferenceList cnsList = cns.resolveToList(null, // sort
            // options
            null, // property names
            new PropertyType[] { PropertyType.PRESENTATION }, // property
            // types
            1);
    if (cnsList.getResolvedConceptReferenceCount() == 0) {
        Util.displayMessage("Unable to find code for CUI = '" + cui + "'");
        return null;
    }
    ResolvedConceptReference rcr = cnsList.getResolvedConceptReference(0);
    Util.displayMessage("Concept code: " + rcr.getConceptCode());
    Util.displayMessage("Description : " + rcr.getEntityDescription().getContent());

    return rcr;
}

/**
 * Recursively display each context that the code participates in with
 * additional information.
 *
 * @param rcr
 * @param lbSvc
 * @param scheme
 * @param csvt
 * @throws LBException
```

```
     */
    protected void printContext(ResolvedConceptReference rcr, LexBIGService lbSvc, String scheme,
            CodingSchemeVersionOrTag csvt) throws LBException {
        Entity node = rcr.getEntity();

        // Step through the presentations and evaluate any tagged with
        // hcd(s) one by one ...
        int hcdsFound = 0;
        for (int i = 0; i < node.getPresentationCount(); i++) {
            Presentation pres = node.getPresentation(i);
            PropertyQualifier[] quals = pres.getPropertyQualifier();
            for (int q = 0; q < quals.length; q++) {
                PropertyQualifier qual = quals[q];
                if (qual.getPropertyQualifierName().equalsIgnoreCase("HCD")) {
                    Text hcd = qual.getValue();
                    Util.displayMessage("");
                    Util.displayMessage("HCD: " + hcd);
                    hcdsFound++;

                    // Define a code set for all concepts tagged with the HCD.
                    CodedNodeSet hcdConceptMatch = lbSvc.getCodingSchemeConcepts(scheme, csvt);
                    hcdConceptMatch.restrictToProperties(null, // property names
                            new PropertyType[] { PropertyType.PRESENTATION }, // property
                            // types
                            null, // source list
                            null, // context list
                            ConvenienceMethods.createNameAndValueList("HCD", hcd.toString()) // qualifier
                            // list
                            );

                    // Define a code graph for all relations tagged with the
                    // HCD.
                    CodedNodeGraph hcdGraph = lbSvc.getNodeGraph(scheme, csvt, "UMLS_Relations");

                    // Now restrict the code graph based on the code set (so,
                    // exclude
                    // nodes that do not contain the desired tag).
                    CodedNodeGraph hcdTargeted = hcdGraph.restrictToTargetCodes(hcdConceptMatch);
                    hcdTargeted.restrictToAssociations(ConvenienceMethods.createNameAndValueList
(hierAssocNames_),
                            ConvenienceMethods.createNameAndValueList("HCD", hcd.toString()));
                    ResolvedConceptReference[] links = hcdTargeted.resolveAsList(null, // graph
                            // focus
                            true, // resolve forward
                            false, // resolve backward
                            Integer.MAX_VALUE, // coded entry depth
                            -1, // association depth
                            null, // property names
                            new PropertyType[] { PropertyType.PRESENTATION }, // property
                            // types
                            null, // sort options
                            500 // max to return
                            ).getResolvedConceptReference();

                    // Resolve and print the chain of terms ...
                    for (int j = 0; j < links.length; j++) {
                        ResolvedConceptReference link = links[j];
                        StringBuffer sb = new StringBuffer(getItemText(link, hcd.toString()));
                        printAssociationList(sb, hcd.toString(), link.getSourceOf(), 0, true);
                        printAssociationList(sb, hcd.toString(), link.getTargetOf(), 0, false);
                        Util.displayMessage(sb.toString());
                    }
                }
            }
        }
        if (hcdsFound == 0) {
            Util.displayMessage("No hierarchical tags (HCD properties) found.");
            return;
        }
    }
```

```
    /**
     * Add text to the given string buffer by recursing the hierarchy and
     * appending text for each included concept.
     *
     * @param sb
     *            StringBuffer to receive the text for the entire list.
     * @param hcd
     *            The hierarchical code to reflect in printed output.
     * @param al
     *            The list to process.
     * @param depth
     *            Current depth of recursion (0=no recursion yet).
     * @param fwd
     *            True if processing relations in a forward direction (source to
     *            target); False otherwise (target to source).
     */
    protected void printAssociationList(StringBuffer sb, String hcd, AssociationList al, int depth, boolean
fwd) {
        if (al != null && al.getAssociationCount() > 0) {
            Association[] aList = al.getAssociation();
            for (int i = 0; i < aList.length; i++) {
                AssociatedConcept[] acList = aList[i].getAssociatedConcepts().getAssociatedConcept();
                for (int j = 0; j < acList.length; j++) {
                    AssociatedConcept ac = acList[j];
                    // Indent on new line according to depth ...
                    sb.append('\n');
                    for (int d = 0; d < depth; d++)
                        sb.append("  ");
                    // Add the code and text for this link ...
                    sb.append("->").append(getItemText(ac, hcd));
                    // Recurse to get the entire chain
                    printAssociationList(sb, hcd, fwd ? ac.getSourceOf() : ac.getTargetOf(), depth + 1, fwd);
                }
            }
        }
    }

    /**
     * Returns the text representing the given item. Note that in general this
     * method assumes the presentations have been constrianed during the initial
     * query by HCD qualification.
     *
     * @param rcr
     *            Concept resolved from a query.
     * @param hcd
     *            The hierarchical code to reflect in printed output.
     * @return The first embedded text presentation, or the item description if
     *         no presentation is found.
     */
    protected String getItemText(ResolvedConceptReference rcr, String hcd) {
        StringBuffer sb = new StringBuffer(rcr.getConceptCode());
        boolean presFound = false;
        if (rcr.getEntity() != null) {
            Presentation[] presentations = rcr.getEntity().getPresentation();
            for (int i = 0; i < presentations.length && !presFound; i++) {
                Presentation p = presentations[i];
                PropertyQualifier[] quals = p.getPropertyQualifier();
                for (int j = 0; j < quals.length && !presFound; j++) {
                    PropertyQualifier pq = quals[j];
                    if (presFound = "HCD".equals(pq.getPropertyQualifierName())
                            && hcd.equals(pq.getValue().getContent())) {
                        sb.append(':').append(p.getValue().getContent());
                    }
                }
            }
        }
        if (!presFound && rcr.getEntityDescription() != null)
            sb.append(':').append(rcr.getEntityDescription().getContent());
        return sb.toString();
    }
}
```