ScoreTerm 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 java.util.Formatter;
import java.util.Iterator;
import java.util.SortedSet;
import java.util.StringTokenizer;
import java.util.TreeSet;

import org.LexGrid.LexBIG.DataModel.Collections.ResolvedConceptReferenceList;
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.CodedNodeSet;
import org.LexGrid.LexBIG.LexBIGService.LexBIGService;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeSet.PropertyType;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeSet.SearchDesignationOption;
import org.LexGrid.LexBIG.Utility.ConvenienceMethods;
import org.LexGrid.LexBIG.Utility.Iterators.ResolvedConceptReferencesIterator;
import org.LexGrid.LexBIG.Utility.LBConstants.MatchAlgorithms;
import org.LexGrid.LexBIG.Utility.LBConstants.SortableProperties;
import org.LexGrid.concepts.Entity;
import org.LexGrid.concepts.Presentation;
import org.apache.commons.collections.BidiMap;
import org.apache.commons.collections.MapIterator;
import org.apache.commons.collections.bidimap.TreeBidiMap;

/**
 * Example showing a simple scoring algorithm that evaluates a provided term
 * against available terms in a code system. A cutoff percentage can optionally
 * be provided.
 */
public class ScoreTerm {

    // Inner class used to manage and sort results.
    protected class ScoredTerm implements Comparable {
        String term;
        float score;

        public int compareTo(Object o) {
            if (o instanceof ScoredTerm) {
                ScoredTerm st = (ScoredTerm) o;
                int i = Float.valueOf(st.score).compareTo(Float.valueOf(score));
                return (i != 0) ? i : term.compareTo(st.term);
            }
```

```
        return 0;
    }

}

public ScoreTerm() {
}

/**
 * Program entry point.
 *
 * @param args
 *            String[]
 */
public static void main(String[] args) {
    if (args.length < 1) {
        System.out.println("Example: ScoreTerm \"some term to evaluate\"");
        System.out.println("Example: ScoreTerm \"some term to evaluate\" 25%");
        return;
    }
    StringBuffer term = new StringBuffer(args[0]);
    float cutoff = 0;
    if (args.length > 1) {
        // Treat everything but last argument automatically as part of
        // search term. This helps in Linux where it is difficult to enter
        // spaces in the command line.
        int lastIndex = args.length - 1;
        for (int i = 1; i < lastIndex; i++)
            term.append(' ').append(args[i]);

        // Treat the last argument as cutoff if numeric, otherwise
        // consider part of the search term as well (same reason as above).
        String val = args[lastIndex].trim();
        try {
            if (val.endsWith("%"))
                val = val.substring(0, val.length() - 1);
            cutoff = Float.valueOf(val);
        } catch (NumberFormatException nfe) {
            term.append(' ').append(val);
        }
    }
    try {
        ScoreTerm pgm = new ScoreTerm();
        pgm.run(term.toString(), cutoff);
    } catch (Exception e) {
        Util.displayAndLogError("REQUEST FAILED !!!", e);
    }
}

/**
 * Runs the score algorithm for a specific term.
 *
 * @param term
 *            The text to evaluate.
 * @param score
 *            Lower cutoff (percentage); a value less than or equal to 0
 *            indicates no cutoff.
 * @throws Exception
 */
public void run(String term, float minScore) throws Exception {
    // Allow the user to pick the target coding scheme.
    // This could also be hardcoded to a specific coding scheme.
    CodingSchemeSummary css = Util.promptForCodeSystem();
    if (css != null) {
        // Determine the set of individual words to compare against.
        SortedSet compareWords = toWords(term);

        // Create a bucket to store results.
        // Sort the results by score (highest score first) and code key.
        BidiMap scoredResult = new TreeBidiMap();
```

```
            // Resolve and iterate through matches to score each.
            // For this example, we keep the highest score per coded concept.
            for (ResolvedConceptReferencesIterator matches = resolveConcepts(css, term); matches.hasNext();) {
                // Work in chunks of 100.
                ResolvedConceptReferenceList refs = matches.next(100);
                for (int i = 0; i < refs.getResolvedConceptReferenceCount(); i++) {
                    ResolvedConceptReference ref = refs.getResolvedConceptReference(i);
                    String code = ref.getConceptCode();

                    Entity node = ref.getEntity();
                    Presentation[] allTermsForConcept = node.getPresentation();

                    for (int j = 0; j < allTermsForConcept.length; j++) {
                        Presentation p = allTermsForConcept[j];
                        String text = p.getValue().getContent();
                        float score = score(toWords(text), compareWords);
                        if (score > minScore) {
                            // Check for a previous match on this code for a
                            // different presentation.
                            // If already present save the item of most
                            // relevance.
                            if (scoredResult.containsKey(code)) {
                                ScoredTerm scoredTerm = (ScoredTerm) scoredResult.get(code);
                                if (scoredTerm.score > score)
                                    continue;
                            }
                            ScoredTerm scoredTerm = new ScoredTerm();
                            scoredTerm.term = text;
                            scoredTerm.score = score;
                            scoredResult.put(code, scoredTerm);
                        }
                    }
                }
            }

            // Print the results.
            printReport(scoredResult);
        }
    }

    /**
     * Resolves matching concepts for any word in the given term.
     *
     * @param css
     *            The code system to search.
     * @param matchWords
     *            The term to match.
     * @return The list of matching references.
     * @throws LBException
     */
    protected ResolvedConceptReferencesIterator resolveConcepts(CodingSchemeSummary css, String query)
            throws LBException {
        // Define a code set over the target terminology and
        // restrict to concepts with matching text based on
        // the provided term.
        LexBIGService lbs = LexBIGServiceImpl.defaultInstance();
        CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
        csvt.setVersion(css.getRepresentsVersion());
        CodedNodeSet cns = lbs.getCodingSchemeConcepts(css.getLocalName(), csvt);

        // Restrict the code set.
        cns
                .restrictToMatchingDesignations(query, SearchDesignationOption.ALL, MatchAlgorithms.LuceneQuery.
name(),
                        null);

        // Resolve the concepts and assigned text.
        ResolvedConceptReferencesIterator matches = cns.resolve(ConvenienceMethods
                .createSortOptionList(new String[] { SortableProperties.code.name() }), null,
                new PropertyType[] { PropertyType.PRESENTATION });
        return matches;
```

```
}

/**
 * Returns a score providing a relative comparison of the first set of words
 * against the second.
 * <p>
 * Currently the score is evaluated as a simple percentage based on number
 * of words in the first set that are also in the second (order
 * independent). This could be enhanced to take order into account, etc.
 *
 * @param wordsToCompare
 * @param wordsToCompareAgainst
 * @return The score (a percentage); a higher value indicates a stronger
 *         match.
 */
protected float score(SortedSet wordsToCompare, SortedSet wordsToCompareAgainst) {
    int totalWords = wordsToCompare.size();
    int matchWords = 0;
    for (Iterator words = wordsToCompare.iterator(); words.hasNext();) {
        String word = words.next().toString();
        if (wordsToCompareAgainst.contains(word))
            matchWords++;
    }
    return ((float) matchWords / (float) totalWords) * 100;
}

/**
 * Display results to the user.
 *
 * @param result
 */
protected void printReport(BidiMap result) {
    final String Dash6 = "------";
    final String Dash10 = "----------";
    final String Dash60 = "------------------------------------------------------------";

    Formatter f = new Formatter();

    // Print header.
    String format = "%-5.5s|%-10.10s|%-60.60s\n";
    Object[] hSep = new Object[] { Dash6, Dash10, Dash60 };
    f.format(format, hSep);
    f.format(format, new Object[] { "Score", "Code", "Term" });
    f.format(format, hSep);

    // Iterate over the result.
    for (MapIterator items = result.inverseBidiMap().mapIterator(); items.hasNext();) {
        ScoredTerm st = (ScoredTerm) items.next();
        String code = (String) items.getValue();

        // Evaluate code
        if (code != null && code.length() > 10)
            code = code.substring(0, 7) + "...";

        // Evaluate term (wrap if necessary)
        String term = st.term;
        if (term != null && term.length() < 60)
            f.format(format, new Object[] { st.score, code, term });
        else {
            String sub = term.substring(0, 60);
            f.format(format, new Object[] { st.score, code, sub });
            int begin = 60;
            int end = term.length();
            while (begin < end) {
                sub = term.substring(begin, Math.min(begin + 60, end));
                f.format(format, new Object[] { "", "", sub });
                begin += 60;
            }
        }
    }
    Util.displayMessage(f.out().toString());
```

```
    }

    /**
     * Return the words comprising the given string, in order ignoring
     * duplicates, common separators and punctuation.
     *
     * @param s
     * @return SortedSet
     */
    @SuppressWarnings("unchecked")
    protected SortedSet toWords(String s) {
        SortedSet words = new TreeSet();
        StringTokenizer st = new StringTokenizer(s, " \t\n\r\f,:+-;");
        while (st.hasMoreTokens())
            words.add(st.nextToken().toLowerCase());
        return words;
    }
}
```