ScoredIterator 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.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.LexGrid.LexBIG.DataModel.Collections.ResolvedConceptReferenceList;
import org.LexGrid.LexBIG.DataModel.Core.ResolvedConceptReference;
import org.LexGrid.LexBIG.Exceptions.LBException;
import org.LexGrid.LexBIG.Exceptions.LBInvocationException;
import org.LexGrid.LexBIG.Exceptions.LBParameterException;
import org.LexGrid.LexBIG.Exceptions.LBResourceUnavailableException;
import org.LexGrid.LexBIG.Utility.Iterators.ResolvedConceptReferencesIterator;

/**
 * Used to wrap scored results for consumption as a standard
 * ResolvedConceptReferenceIterator.
 */
public class ScoredIterator implements ResolvedConceptReferencesIterator {
    private static final long serialVersionUID = -4975979409144702081L;
    ScoredTerm[] scoredTerms = null;
    int position = 0;
    int scrolled = 0;

    /**
     * Construct from a map of ScoredTerms, ordered from high to low score. If >
     * 0, the maximum items parameter sets an upper limit on the number of
     * top-scored items to maintain and return.
     *
     * @param scoredTerms
     * @param maxItems
     */
    public ScoredIterator(Collection<ScoredTerm> scoredTerms, int maxItems) {
        // Add all scored terms, sorted according to ScoredTerm comparator ...
        List<ScoredTerm> temp = new ArrayList<ScoredTerm>();
        temp.addAll(scoredTerms);
        Collections.sort(temp);

        // Maintain only as many items as the specified maximum ...
        int limit = maxItems > 0 ? Math.min(maxItems, scoredTerms.size()) : scoredTerms.size();
        int count = 0;
        this.scoredTerms = new ScoredTerm[limit];
        for (Iterator<ScoredTerm> terms = temp.listIterator(); terms.hasNext() && count < limit; count++)
            this.scoredTerms[count] = terms.next();
```

```
}

/**
 * Construct from a pre-sorted array of ScoredTerms.
 *
 * @param scoredTerms
 */
public ScoredIterator(ScoredTerm[] scoredTerms) {
    this.scoredTerms = scoredTerms;
}

/**
 * Returns a specific range of items without altering cursor position.
 */
public ResolvedConceptReferenceList get(int start, int end) throws LBResourceUnavailableException,
        LBInvocationException, LBParameterException {
    verifyResources();
    ResolvedConceptReferenceList result = new ResolvedConceptReferenceList();
    int stop = Math.max(0, Math.min(scoredTerms.length, end));
    if (start < 0 || stop < start)
        throw new LBParameterException("Index out of bounds.");
    for (int i = start; i < stop; i++)
        result.addResolvedConceptReference(scoredTerms[i].ref);
    return result;
}

/**
 * Returns items skipped by last scroll() without altering cursor position.
 */
public ResolvedConceptReferenceList getNext() {
    ResolvedConceptReferenceList result = null;
    try {
        result = get(position - scrolled, position);
    } catch (LBException e) {
        result = new ResolvedConceptReferenceList();
        e.printStackTrace();
    }
    return result;
}

/**
 * Returns the next item and advances the cursor.
 */
public ResolvedConceptReference next() throws LBResourceUnavailableException, LBInvocationException {
    verifyResources();
    return hasNext() ? scoredTerms[position++].ref : null;
}

/**
 * Returns the next 'n' items and advances the cursor.
 */
public ResolvedConceptReferenceList next(int maxToReturn) throws LBResourceUnavailableException,
        LBInvocationException {
    verifyResources();
    ResolvedConceptReferenceList result = null;
    try {
        int pageSize = Math.max(0, Math.min(maxToReturn, numberRemaining()));
        result = get(position, position + pageSize);
        position += pageSize;
    } catch (LBParameterException e) {
        result = new ResolvedConceptReferenceList();
        e.printStackTrace();
    }
    return result;
}

/**
 * Skips 'n' items which are available via getNext() until a call to next(),
 * returning self.
 */
public ResolvedConceptReferencesIterator scroll(int maxToReturn) throws LBResourceUnavailableException,
```

```
            LBInvocationException {
        verifyResources();
        scrolled = Math.max(0, Math.min(maxToReturn, numberRemaining()));
        position += scrolled;
        return this;
    }

    /**
     * Indicates if more items are available through next() operations.
     */
    public boolean hasNext() throws LBResourceUnavailableException {
        verifyResources();
        return position < scoredTerms.length;
    }

    /**
     * Indicates the number of items available to retrieve via next()
     * operations.
     */
    public int numberRemaining() throws LBResourceUnavailableException {
        return hasNext() ? scoredTerms.length - position : 0;
    }

    /**
     * Releases the maintained terms and invalidates the iterator.
     */
    public void release() throws LBResourceUnavailableException {
        scoredTerms = null;
    }

    /**
     * Verifies the iterator is still valid.
     */
    protected void verifyResources() throws LBResourceUnavailableException {
        if (scoredTerms == null)
            throw new LBResourceUnavailableException("Iterator resources released.");
    }
}
```