001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017package org.apache.commons.collections4;  
018  
019import java.util.ArrayList;  
020import java.util.Enumeration;  
021import java.util.List;  
022import java.util.StringTokenizer;  
023  
024import org.apache.commons.collections4.iterators.EnumerationIterator;  
025  
026/\*\*  
027 \* Provides utility methods for {@link Enumeration} instances.  
028 \*  
029 \* @since 3.0  
030 \*/  
031public class EnumerationUtils {  
032  
033 /\*\*  
034 \* EnumerationUtils is not normally instantiated.  
035 \*/  
036 private EnumerationUtils() {}  
037  
038 /\*\*  
039 \* Returns the <code>index</code>-th value in the {@link Enumeration}, throwing  
040 \* <code>IndexOutOfBoundsException</code> if there is no such element.  
041 \* <p>  
042 \* The Enumeration is advanced to <code>index</code> (or to the end, if  
043 \* <code>index</code> exceeds the number of entries) as a side effect of this method.  
044 \*  
045 \* @param e the enumeration to get a value from  
046 \* @param index the index to get  
047 \* @param <T> the type of object in the {@link Enumeration}  
048 \* @return the object at the specified index  
049 \* @throws IndexOutOfBoundsException if the index is invalid  
050 \* @throws IllegalArgumentException if the object type is invalid  
051 \* @since 4.1  
052 \*/  
053 public static <T> T get(final Enumeration<T> e, final int index) {  
054 int i = index;  
055 CollectionUtils.checkIndexBounds(i);  
056 while (e.hasMoreElements()) {  
057 i--;  
058 if (i == -1) {  
059 return e.nextElement();  
060 }  
061 e.nextElement();  
062 }  
063 throw new IndexOutOfBoundsException("Entry does not exist: " + i);  
064 }  
065  
066 /\*\*  
067 \* Creates a list based on an enumeration.  
068 \*  
069 \* <p>As the enumeration is traversed, an ArrayList of its values is  
070 \* created. The new list is returned.</p>  
071 \*  
072 \* @param <E> the element type  
073 \* @param enumeration the enumeration to traverse, which should not be <code>null</code>.  
074 \* @return a list containing all elements of the given enumeration  
075 \* @throws NullPointerException if the enumeration parameter is <code>null</code>.  
076 \*/  
077 public static <E> List<E> toList(final Enumeration<? extends E> enumeration) {  
078 return IteratorUtils.toList(new EnumerationIterator<>(enumeration));  
079 }  
080  
081 /\*\*  
082 \* Override toList(Enumeration) for StringTokenizer as it implements Enumeration<Object>  
083 \* for the sake of backward compatibility.  
084 \*  
085 \* @param stringTokenizer the tokenizer to convert to a {@link List}<{@link String}>  
086 \* @return a list containing all tokens of the given StringTokenizer  
087 \*/  
088 public static List<String> toList(final StringTokenizer stringTokenizer) {  
089 final List<String> result = new ArrayList<>(stringTokenizer.countTokens());  
090 while (stringTokenizer.hasMoreTokens()) {  
091 result.add(stringTokenizer.nextToken());  
092 }  
093 return result;  
094 }  
095}