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.iterators;  
018  
019import java.util.Collection;  
020import java.util.Enumeration;  
021import java.util.Iterator;  
022  
023/\*\*  
024 \* Adapter to make {@link Enumeration Enumeration} instances appear  
025 \* to be {@link Iterator Iterator} instances.  
026 \*  
027 \* @since 1.0  
028 \*/  
029public class EnumerationIterator<E> implements Iterator<E> {  
030  
031 /\*\* The collection to remove elements from \*/  
032 private final Collection<? super E> collection;  
033 /\*\* The enumeration being converted \*/  
034 private Enumeration<? extends E> enumeration;  
035 /\*\* The last object retrieved \*/  
036 private E last;  
037  
038 // Constructors  
039 //-----------------------------------------------------------------------  
040 /\*\*  
041 \* Constructs a new <code>EnumerationIterator</code> that will not  
042 \* function until {@link #setEnumeration(Enumeration)} is called.  
043 \*/  
044 public EnumerationIterator() {  
045 this(null, null);  
046 }  
047  
048 /\*\*  
049 \* Constructs a new <code>EnumerationIterator</code> that provides  
050 \* an iterator view of the given enumeration.  
051 \*  
052 \* @param enumeration the enumeration to use  
053 \*/  
054 public EnumerationIterator(final Enumeration<? extends E> enumeration) {  
055 this(enumeration, null);  
056 }  
057  
058 /\*\*  
059 \* Constructs a new <code>EnumerationIterator</code> that will remove  
060 \* elements from the specified collection.  
061 \*  
062 \* @param enumeration the enumeration to use  
063 \* @param collection the collection to remove elements from  
064 \*/  
065 public EnumerationIterator(final Enumeration<? extends E> enumeration, final Collection<? super E> collection) {  
066 super();  
067 this.enumeration = enumeration;  
068 this.collection = collection;  
069 this.last = null;  
070 }  
071  
072 // Iterator interface  
073 //-----------------------------------------------------------------------  
074 /\*\*  
075 \* Returns true if the underlying enumeration has more elements.  
076 \*  
077 \* @return true if the underlying enumeration has more elements  
078 \* @throws NullPointerException if the underlying enumeration is null  
079 \*/  
080 @Override  
081 public boolean hasNext() {  
082 return enumeration.hasMoreElements();  
083 }  
084  
085 /\*\*  
086 \* Returns the next object from the enumeration.  
087 \*  
088 \* @return the next object from the enumeration  
089 \* @throws NullPointerException if the enumeration is null  
090 \*/  
091 @Override  
092 public E next() {  
093 last = enumeration.nextElement();  
094 return last;  
095 }  
096  
097 /\*\*  
098 \* Removes the last retrieved element if a collection is attached.  
099 \* <p>  
100 \* Functions if an associated <code>Collection</code> is known.  
101 \* If so, the first occurrence of the last returned object from this  
102 \* iterator will be removed from the collection.  
103 \*  
104 \* @throws IllegalStateException <code>next()</code> not called.  
105 \* @throws UnsupportedOperationException if no associated collection  
106 \*/  
107 @Override  
108 public void remove() {  
109 if (collection != null) {  
110 if (last != null) {  
111 collection.remove(last);  
112 } else {  
113 throw new IllegalStateException("next() must have been called for remove() to function");  
114 }  
115 } else {  
116 throw new UnsupportedOperationException("No Collection associated with this Iterator");  
117 }  
118 }  
119  
120 // Properties  
121 //-----------------------------------------------------------------------  
122 /\*\*  
123 \* Returns the underlying enumeration.  
124 \*  
125 \* @return the underlying enumeration  
126 \*/  
127 public Enumeration<? extends E> getEnumeration() {  
128 return enumeration;  
129 }  
130  
131 /\*\*  
132 \* Sets the underlying enumeration.  
133 \*  
134 \* @param enumeration the new underlying enumeration  
135 \*/  
136 public void setEnumeration(final Enumeration<? extends E> enumeration) {  
137 this.enumeration = enumeration;  
138 }  
139  
140}