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.Iterator;  
021import java.util.NoSuchElementException;  
022  
023import org.apache.commons.collections4.ResettableIterator;  
024  
025/\*\*  
026 \* An Iterator that restarts when it reaches the end.  
027 \* <p>  
028 \* The iterator will loop continuously around the provided elements, unless  
029 \* there are no elements in the collection to begin with, or all the elements  
030 \* have been {@link #remove removed}.  
031 \* <p>  
032 \* Concurrent modifications are not directly supported, and for most collection  
033 \* implementations will throw a ConcurrentModificationException.  
034 \*  
035 \* @since 3.0  
036 \*/  
037public class LoopingIterator<E> implements ResettableIterator<E> {  
038  
039 /\*\* The collection to base the iterator on \*/  
040 private final Collection<? extends E> collection;  
041 /\*\* The current iterator \*/  
042 private Iterator<? extends E> iterator;  
043  
044 /\*\*  
045 \* Constructor that wraps a collection.  
046 \* <p>  
047 \* There is no way to reset an Iterator instance without recreating it from  
048 \* the original source, so the Collection must be passed in.  
049 \*  
050 \* @param coll the collection to wrap  
051 \* @throws NullPointerException if the collection is null  
052 \*/  
053 public LoopingIterator(final Collection<? extends E> coll) {  
054 if (coll == null) {  
055 throw new NullPointerException("The collection must not be null");  
056 }  
057 collection = coll;  
058 reset();  
059 }  
060  
061 /\*\*  
062 \* Has the iterator any more elements.  
063 \* <p>  
064 \* Returns false only if the collection originally had zero elements, or  
065 \* all the elements have been {@link #remove removed}.  
066 \*  
067 \* @return <code>true</code> if there are more elements  
068 \*/  
069 @Override  
070 public boolean hasNext() {  
071 return collection.size() > 0;  
072 }  
073  
074 /\*\*  
075 \* Returns the next object in the collection.  
076 \* <p>  
077 \* If at the end of the collection, return the first element.  
078 \*  
079 \* @return the next object  
080 \* @throws NoSuchElementException if there are no elements  
081 \* at all. Use {@link #hasNext} to avoid this error.  
082 \*/  
083 @Override  
084 public E next() {  
085 if (collection.size() == 0) {  
086 throw new NoSuchElementException("There are no elements for this iterator to loop on");  
087 }  
088 if (iterator.hasNext() == false) {  
089 reset();  
090 }  
091 return iterator.next();  
092 }  
093  
094 /\*\*  
095 \* Removes the previously retrieved item from the underlying collection.  
096 \* <p>  
097 \* This feature is only supported if the underlying collection's  
098 \* {@link Collection#iterator iterator} method returns an implementation  
099 \* that supports it.  
100 \* <p>  
101 \* This method can only be called after at least one {@link #next} method call.  
102 \* After a removal, the remove method may not be called again until another  
103 \* next has been performed. If the {@link #reset} is called, then remove may  
104 \* not be called until {@link #next} is called again.  
105 \*/  
106 @Override  
107 public void remove() {  
108 iterator.remove();  
109 }  
110  
111 /\*\*  
112 \* Resets the iterator back to the start of the collection.  
113 \*/  
114 @Override  
115 public void reset() {  
116 iterator = collection.iterator();  
117 }  
118  
119 /\*\*  
120 \* Gets the size of the collection underlying the iterator.  
121 \*  
122 \* @return the current collection size  
123 \*/  
124 public int size() {  
125 return collection.size();  
126 }  
127  
128}