001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with this  
004 \* work for additional information regarding copyright ownership. The ASF  
005 \* licenses this file to You under the Apache License, Version 2.0 (the  
006 \* "License"); you may not use this file except in compliance with the License.  
007 \* You may obtain a copy of the License at  
008 \* http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law  
009 \* or agreed to in writing, software distributed under the License is  
010 \* distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY  
011 \* KIND, either express or implied. See the License for the specific language  
012 \* governing permissions and limitations under the License.  
013 \*/  
014package org.apache.commons.collections4.iterators;  
015  
016import java.util.Iterator;  
017  
018import org.apache.commons.collections4.ResettableIterator;  
019  
020/\*\*  
021 \* Adapter to make an {@link Iterator Iterator} instance appear to be an  
022 \* {@link Iterable Iterable} instance. The iterable can be constructed in one  
023 \* of two variants: single use, multiple use.  
024 \*  
025 \* <p>  
026 \* In the single use iterable case, the iterable is only usable for one  
027 \* iterative operation over the source iterator. Subsequent iterative  
028 \* operations use the same, exhausted source iterator. To create a single use  
029 \* iterable, construct a new {@link IteratorIterable} using a {@link Iterator}  
030 \* that is NOT a {@link ResettableIterator} iterator:  
031 \* </p>  
032 \*  
033 \* <pre>  
034 \* Iterator<Integer> iterator = // some non-resettable iterator  
035 \* Iterable<Integer> iterable = new IteratorIterable<Integer>(iterator);  
036 \* </pre>  
037 \*  
038 \* <p>  
039 \* In the multiple use iterable case, the iterable is usable for any number of  
040 \* iterative operations over the source iterator. Of special note, even though  
041 \* the iterable supports multiple iterations, it does not support concurrent  
042 \* iterations. To implicitly create a multiple use iterable, construct a new  
043 \* {@link IteratorIterable} using a {@link ResettableIterator} iterator:  
044 \* </p>  
045 \*  
046 \* <pre>  
047 \* Integer[] array = {Integer.valueOf(1),Integer.valueOf(2),Integer.valueOf(3)};  
048 \* Iterator<Integer> iterator = IteratorUtils.arrayIterator(array); // a resettable iterator  
049 \* Iterable<Integer> iterable = new IteratorIterable<Integer>(iterator);  
050 \* </pre>  
051 \*  
052 \* <p>  
053 \* A multiple use iterable can also be explicitly constructed using any  
054 \* {@link Iterator} and specifying <code>true</code> for the  
055 \* <code>multipleUse</code> flag:  
056 \* </p>  
057 \*  
058 \* <pre>  
059 \* Iterator<Integer> iterator = // some non-resettable iterator  
060 \* Iterable<Integer> iterable = new IteratorIterable<Integer>(iterator, true);  
061 \* </pre>  
062 \*  
063 \* @since 4.0  
064 \*/  
065public class IteratorIterable<E> implements Iterable<E> {  
066  
067 /\*\*  
068 \* Factory method to create an {@link Iterator Iterator} from another  
069 \* iterator over objects of a different subtype.  
070 \*/  
071 private static <E> Iterator<E> createTypesafeIterator(final Iterator<? extends E> iterator) {  
072 return new Iterator<E>() {  
073 @Override  
074 public boolean hasNext() {  
075 return iterator.hasNext();  
076 }  
077  
078 @Override  
079 public E next() {  
080 return iterator.next();  
081 }  
082  
083 @Override  
084 public void remove() {  
085 iterator.remove();  
086 }  
087 };  
088 }  
089  
090 /\*\* the iterator being adapted into an iterable. \*/  
091 private final Iterator<? extends E> iterator;  
092  
093 /\*\* the iterator parameterized as the {@link #iterator()} return type. \*/  
094 private final Iterator<E> typeSafeIterator;  
095  
096 /\*\*  
097 \* Constructs a new <code>IteratorIterable</code> that will use the given  
098 \* iterator.  
099 \*  
100 \* @param iterator the iterator to use.  
101 \*/  
102 public IteratorIterable(final Iterator<? extends E> iterator) {  
103 this(iterator, false);  
104 }  
105  
106 /\*\*  
107 \* Constructs a new <code>IteratorIterable</code> that will use the given  
108 \* iterator.  
109 \*  
110 \* @param iterator the iterator to use.  
111 \* @param multipleUse <code>true</code> if the new iterable can be used in multiple iterations  
112 \*/  
113 public IteratorIterable(final Iterator<? extends E> iterator, final boolean multipleUse) {  
114 super();  
115 if (multipleUse && !(iterator instanceof ResettableIterator)) {  
116 this.iterator = new ListIteratorWrapper<>(iterator);  
117 } else {  
118 this.iterator = iterator;  
119 }  
120 this.typeSafeIterator = createTypesafeIterator(this.iterator);  
121 }  
122  
123 /\*\*  
124 \* Gets the iterator wrapped by this iterable.  
125 \*  
126 \* @return the iterator  
127 \*/  
128 @Override  
129 public Iterator<E> iterator() {  
130 if (iterator instanceof ResettableIterator) {  
131 ((ResettableIterator<? extends E>)iterator).reset();  
132 }  
133 return typeSafeIterator;  
134 }  
135}