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.List;  
020import java.util.ListIterator;  
021  
022import org.apache.commons.collections4.ResettableListIterator;  
023  
024/\*\*  
025 \* Iterates backwards through a List, starting with the last element  
026 \* and continuing to the first. This is useful for looping around  
027 \* a list in reverse order without needing to actually reverse the list.  
028 \* <p>  
029 \* The first call to <code>next()</code> will return the last element  
030 \* from the list, and so on. The <code>hasNext()</code> method works  
031 \* in concert with the <code>next()</code> method as expected.  
032 \* However, the <code>nextIndex()</code> method returns the correct  
033 \* index in the list, thus it starts high and reduces as the iteration  
034 \* continues. The previous methods work similarly.  
035 \*  
036 \* @since 3.2  
037 \*/  
038public class ReverseListIterator<E> implements ResettableListIterator<E> {  
039  
040 /\*\* The list being wrapped. \*/  
041 private final List<E> list;  
042 /\*\* The list iterator being wrapped. \*/  
043 private ListIterator<E> iterator;  
044 /\*\* Flag to indicate if updating is possible at the moment. \*/  
045 private boolean validForUpdate = true;  
046  
047 /\*\*  
048 \* Constructor that wraps a list.  
049 \*  
050 \* @param list the list to create a reversed iterator for  
051 \* @throws NullPointerException if the list is null  
052 \*/  
053 public ReverseListIterator(final List<E> list) {  
054 super();  
055 if (list == null) {  
056 throw new NullPointerException("List must not be null.");  
057 }  
058 this.list = list;  
059 iterator = list.listIterator(list.size());  
060 }  
061  
062 //-----------------------------------------------------------------------  
063 /\*\*  
064 \* Checks whether there is another element.  
065 \*  
066 \* @return true if there is another element  
067 \*/  
068 @Override  
069 public boolean hasNext() {  
070 return iterator.hasPrevious();  
071 }  
072  
073 /\*\*  
074 \* Gets the next element.  
075 \* The next element is the previous in the list.  
076 \*  
077 \* @return the next element in the iterator  
078 \*/  
079 @Override  
080 public E next() {  
081 final E obj = iterator.previous();  
082 validForUpdate = true;  
083 return obj;  
084 }  
085  
086 /\*\*  
087 \* Gets the index of the next element.  
088 \*  
089 \* @return the index of the next element in the iterator  
090 \*/  
091 @Override  
092 public int nextIndex() {  
093 return iterator.previousIndex();  
094 }  
095  
096 /\*\*  
097 \* Checks whether there is a previous element.  
098 \*  
099 \* @return true if there is a previous element  
100 \*/  
101 @Override  
102 public boolean hasPrevious() {  
103 return iterator.hasNext();  
104 }  
105  
106 /\*\*  
107 \* Gets the previous element.  
108 \* The next element is the previous in the list.  
109 \*  
110 \* @return the previous element in the iterator  
111 \*/  
112 @Override  
113 public E previous() {  
114 final E obj = iterator.next();  
115 validForUpdate = true;  
116 return obj;  
117 }  
118  
119 /\*\*  
120 \* Gets the index of the previous element.  
121 \*  
122 \* @return the index of the previous element in the iterator  
123 \*/  
124 @Override  
125 public int previousIndex() {  
126 return iterator.nextIndex();  
127 }  
128  
129 /\*\*  
130 \* Removes the last returned element.  
131 \*  
132 \* @throws UnsupportedOperationException if the list is unmodifiable  
133 \* @throws IllegalStateException if there is no element to remove  
134 \*/  
135 @Override  
136 public void remove() {  
137 if (validForUpdate == false) {  
138 throw new IllegalStateException("Cannot remove from list until next() or previous() called");  
139 }  
140 iterator.remove();  
141 }  
142  
143 /\*\*  
144 \* Replaces the last returned element.  
145 \*  
146 \* @param obj the object to set  
147 \* @throws UnsupportedOperationException if the list is unmodifiable  
148 \* @throws IllegalStateException if the iterator is not in a valid state for set  
149 \*/  
150 @Override  
151 public void set(final E obj) {  
152 if (validForUpdate == false) {  
153 throw new IllegalStateException("Cannot set to list until next() or previous() called");  
154 }  
155 iterator.set(obj);  
156 }  
157  
158 /\*\*  
159 \* Adds a new element to the list between the next and previous elements.  
160 \*  
161 \* @param obj the object to add  
162 \* @throws UnsupportedOperationException if the list is unmodifiable  
163 \* @throws IllegalStateException if the iterator is not in a valid state for set  
164 \*/  
165 @Override  
166 public void add(final E obj) {  
167 // the validForUpdate flag is needed as the necessary previous()  
168 // method call re-enables remove and add  
169 if (validForUpdate == false) {  
170 throw new IllegalStateException("Cannot add to list until next() or previous() called");  
171 }  
172 validForUpdate = false;  
173 iterator.add(obj);  
174 iterator.previous();  
175 }  
176  
177 /\*\*  
178 \* Resets the iterator back to the start (which is the  
179 \* end of the list as this is a reversed iterator)  
180 \*/  
181 @Override  
182 public void reset() {  
183 iterator = list.listIterator(list.size());  
184 }  
185  
186}