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.NoSuchElementException;  
020  
021import org.apache.commons.collections4.ResettableListIterator;  
022  
023/\*\*  
024 \* <code>SingletonIterator</code> is an {@link java.util.ListIterator} over a single  
025 \* object instance.  
026 \*  
027 \* @since 2.1  
028 \*/  
029public class SingletonListIterator<E> implements ResettableListIterator<E> {  
030  
031 private boolean beforeFirst = true;  
032 private boolean nextCalled = false;  
033 private boolean removed = false;  
034 private E object;  
035  
036 /\*\*  
037 \* Constructs a new <code>SingletonListIterator</code>.  
038 \*  
039 \* @param object the single object to return from the iterator  
040 \*/  
041 public SingletonListIterator(final E object) {  
042 super();  
043 this.object = object;  
044 }  
045  
046 /\*\*  
047 \* Is another object available from the iterator?  
048 \* <p>  
049 \* This returns true if the single object hasn't been returned yet.  
050 \*  
051 \* @return true if the single object hasn't been returned yet  
052 \*/  
053 @Override  
054 public boolean hasNext() {  
055 return beforeFirst && !removed;  
056 }  
057  
058 /\*\*  
059 \* Is a previous object available from the iterator?  
060 \* <p>  
061 \* This returns true if the single object has been returned.  
062 \*  
063 \* @return true if the single object has been returned  
064 \*/  
065 @Override  
066 public boolean hasPrevious() {  
067 return !beforeFirst && !removed;  
068 }  
069  
070 /\*\*  
071 \* Returns the index of the element that would be returned by a subsequent  
072 \* call to {@code next}.  
073 \*  
074 \* @return 0 or 1 depending on current state.  
075 \*/  
076 @Override  
077 public int nextIndex() {  
078 return beforeFirst ? 0 : 1;  
079 }  
080  
081 /\*\*  
082 \* Returns the index of the element that would be returned by a subsequent  
083 \* call to {@code previous}. A return value of -1 indicates that the iterator is currently at  
084 \* the start.  
085 \*  
086 \* @return 0 or -1 depending on current state.  
087 \*/  
088 @Override  
089 public int previousIndex() {  
090 return beforeFirst ? -1 : 0;  
091 }  
092  
093 /\*\*  
094 \* Get the next object from the iterator.  
095 \* <p>  
096 \* This returns the single object if it hasn't been returned yet.  
097 \*  
098 \* @return the single object  
099 \* @throws NoSuchElementException if the single object has already  
100 \* been returned  
101 \*/  
102 @Override  
103 public E next() {  
104 if (!beforeFirst || removed) {  
105 throw new NoSuchElementException();  
106 }  
107 beforeFirst = false;  
108 nextCalled = true;  
109 return object;  
110 }  
111  
112 /\*\*  
113 \* Get the previous object from the iterator.  
114 \* <p>  
115 \* This returns the single object if it has been returned.  
116 \*  
117 \* @return the single object  
118 \* @throws NoSuchElementException if the single object has not already  
119 \* been returned  
120 \*/  
121 @Override  
122 public E previous() {  
123 if (beforeFirst || removed) {  
124 throw new NoSuchElementException();  
125 }  
126 beforeFirst = true;  
127 return object;  
128 }  
129  
130 /\*\*  
131 \* Remove the object from this iterator.  
132 \* @throws IllegalStateException if the {@code next} or {@code previous}  
133 \* method has not yet been called, or the {@code remove} method  
134 \* has already been called after the last call to {@code next}  
135 \* or {@code previous}.  
136 \*/  
137 @Override  
138 public void remove() {  
139 if(!nextCalled || removed) {  
140 throw new IllegalStateException();  
141 }  
142 object = null;  
143 removed = true;  
144 }  
145  
146 /\*\*  
147 \* Add always throws {@link UnsupportedOperationException}.  
148 \*  
149 \* @param obj the object to add  
150 \* @throws UnsupportedOperationException always  
151 \*/  
152 @Override  
153 public void add(final E obj) {  
154 throw new UnsupportedOperationException("add() is not supported by this iterator");  
155 }  
156  
157 /\*\*  
158 \* Set sets the value of the singleton.  
159 \*  
160 \* @param obj the object to set  
161 \* @throws IllegalStateException if {@code next} has not been called  
162 \* or the object has been removed  
163 \*/  
164 @Override  
165 public void set(final E obj) {  
166 if (!nextCalled || removed) {  
167 throw new IllegalStateException();  
168 }  
169 this.object = obj;  
170 }  
171  
172 /\*\*  
173 \* Reset the iterator back to the start.  
174 \*/  
175 @Override  
176 public void reset() {  
177 beforeFirst = true;  
178 nextCalled = false;  
179 }  
180  
181}