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.lang.reflect.Array;  
020import java.util.NoSuchElementException;  
021  
022import org.apache.commons.collections4.ResettableListIterator;  
023  
024/\*\*  
025 \* Implements a {@link java.util.ListIterator} over an array.  
026 \* <p>  
027 \* The array can be either an array of object or of primitives. If you know  
028 \* that you have an object array, the {@link ObjectArrayListIterator}  
029 \* class is a better choice, as it will perform better.  
030 \*  
031 \* <p>  
032 \* This iterator does not support {@link #add(Object)} or {@link #remove()}, as the array  
033 \* cannot be changed in size. The {@link #set(Object)} method is supported however.  
034 \*  
035 \* @see org.apache.commons.collections4.iterators.ArrayIterator  
036 \* @see java.util.Iterator  
037 \* @see java.util.ListIterator  
038 \*  
039 \* @since 3.0  
040 \*/  
041public class ArrayListIterator<E> extends ArrayIterator<E>  
042 implements ResettableListIterator<E> {  
043  
044 /\*\*  
045 \* Holds the index of the last item returned by a call to <code>next()</code>  
046 \* or <code>previous()</code>. This is set to <code>-1</code> if neither method  
047 \* has yet been invoked. <code>lastItemIndex</code> is used to to implement  
048 \* the {@link #set} method.  
049 \*/  
050 private int lastItemIndex = -1;  
051  
052 // Constructors  
053 // ----------------------------------------------------------------------  
054 /\*\*  
055 \* Constructs an ArrayListIterator that will iterate over the values in the  
056 \* specified array.  
057 \*  
058 \* @param array the array to iterate over  
059 \* @throws IllegalArgumentException if <code>array</code> is not an array.  
060 \* @throws NullPointerException if <code>array</code> is <code>null</code>  
061 \*/  
062 public ArrayListIterator(final Object array) {  
063 super(array);  
064 }  
065  
066 /\*\*  
067 \* Constructs an ArrayListIterator that will iterate over the values in the  
068 \* specified array from a specific start index.  
069 \*  
070 \* @param array the array to iterate over  
071 \* @param startIndex the index to start iterating at  
072 \* @throws IllegalArgumentException if <code>array</code> is not an array.  
073 \* @throws NullPointerException if <code>array</code> is <code>null</code>  
074 \* @throws IndexOutOfBoundsException if the start index is out of bounds  
075 \*/  
076 public ArrayListIterator(final Object array, final int startIndex) {  
077 super(array, startIndex);  
078 }  
079  
080 /\*\*  
081 \* Construct an ArrayListIterator that will iterate over a range of values  
082 \* in the specified array.  
083 \*  
084 \* @param array the array to iterate over  
085 \* @param startIndex the index to start iterating at  
086 \* @param endIndex the index (exclusive) to finish iterating at  
087 \* @throws IllegalArgumentException if <code>array</code> is not an array.  
088 \* @throws IndexOutOfBoundsException if the start or end index is out of bounds  
089 \* @throws IllegalArgumentException if end index is before the start  
090 \* @throws NullPointerException if <code>array</code> is <code>null</code>  
091 \*/  
092 public ArrayListIterator(final Object array, final int startIndex, final int endIndex) {  
093 super(array, startIndex, endIndex);  
094 }  
095  
096 // ListIterator interface  
097 //-----------------------------------------------------------------------  
098 /\*\*  
099 \* Returns true if there are previous elements to return from the array.  
100 \*  
101 \* @return true if there is a previous element to return  
102 \*/  
103 @Override  
104 public boolean hasPrevious() {  
105 return this.index > this.startIndex;  
106 }  
107  
108 /\*\*  
109 \* Gets the previous element from the array.  
110 \*  
111 \* @return the previous element  
112 \* @throws NoSuchElementException if there is no previous element  
113 \*/  
114 @Override  
115 @SuppressWarnings("unchecked")  
116 public E previous() {  
117 if (hasPrevious() == false) {  
118 throw new NoSuchElementException();  
119 }  
120 this.lastItemIndex = --this.index;  
121 return (E) Array.get(this.array, this.index);  
122 }  
123  
124 /\*\*  
125 \* Gets the next element from the array.  
126 \*  
127 \* @return the next element  
128 \* @throws NoSuchElementException if there is no next element  
129 \*/  
130 @Override  
131 @SuppressWarnings("unchecked")  
132 public E next() {  
133 if (hasNext() == false) {  
134 throw new NoSuchElementException();  
135 }  
136 this.lastItemIndex = this.index;  
137 return (E) Array.get(this.array, this.index++);  
138 }  
139  
140 /\*\*  
141 \* Gets the next index to be retrieved.  
142 \*  
143 \* @return the index of the item to be retrieved next  
144 \*/  
145 @Override  
146 public int nextIndex() {  
147 return this.index - this.startIndex;  
148 }  
149  
150 /\*\*  
151 \* Gets the index of the item to be retrieved if {@link #previous()} is called.  
152 \*  
153 \* @return the index of the item to be retrieved next  
154 \*/  
155 @Override  
156 public int previousIndex() {  
157 return this.index - this.startIndex - 1;  
158 }  
159  
160 /\*\*  
161 \* This iterator does not support modification of its backing collection, and so will  
162 \* always throw an {@link UnsupportedOperationException} when this method is invoked.  
163 \*  
164 \* @param o the element to add  
165 \* @throws UnsupportedOperationException always thrown.  
166 \* @see java.util.ListIterator#set  
167 \*/  
168 @Override  
169 public void add(final Object o) {  
170 throw new UnsupportedOperationException("add() method is not supported");  
171 }  
172  
173 /\*\*  
174 \* Sets the element under the cursor.  
175 \* <p>  
176 \* This method sets the element that was returned by the last call  
177 \* to {@link #next()} of {@link #previous()}.  
178 \* <p>  
179 \* <b>Note:</b> {@link java.util.ListIterator} implementations that support  
180 \* <code>add()</code> and <code>remove()</code> only allow <code>set()</code> to be called  
181 \* once per call to <code>next()</code> or <code>previous</code> (see the {@link java.util.ListIterator}  
182 \* javadoc for more details). Since this implementation does  
183 \* not support <code>add()</code> or <code>remove()</code>, <code>set()</code> may be  
184 \* called as often as desired.  
185 \*  
186 \* @param o the element to set  
187 \* @throws IllegalStateException if {@link #next()} or {@link #previous()} has not been called  
188 \* before {@link #set(Object)}  
189 \* @see java.util.ListIterator#set  
190 \*/  
191 @Override  
192 public void set(final Object o) {  
193 if (this.lastItemIndex == -1) {  
194 throw new IllegalStateException("must call next() or previous() before a call to set()");  
195 }  
196  
197 Array.set(this.array, this.lastItemIndex, o);  
198 }  
199  
200 /\*\*  
201 \* Resets the iterator back to the start index.  
202 \*/  
203 @Override  
204 public void reset() {  
205 super.reset();  
206 this.lastItemIndex = -1;  
207 }  
208  
209}