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.ListIterator;  
020  
021/\*\*  
022 \* Provides basic behaviour for decorating a list iterator with extra functionality.  
023 \* <p>  
024 \* All methods are forwarded to the decorated list iterator.  
025 \*  
026 \* @since 3.0  
027 \*/  
028public class AbstractListIteratorDecorator<E> implements ListIterator<E> {  
029  
030 /\*\* The iterator being decorated \*/  
031 private final ListIterator<E> iterator;  
032  
033 //-----------------------------------------------------------------------  
034 /\*\*  
035 \* Constructor that decorates the specified iterator.  
036 \*  
037 \* @param iterator the iterator to decorate, must not be null  
038 \* @throws NullPointerException if the iterator is null  
039 \*/  
040 public AbstractListIteratorDecorator(final ListIterator<E> iterator) {  
041 super();  
042 if (iterator == null) {  
043 throw new NullPointerException("ListIterator must not be null");  
044 }  
045 this.iterator = iterator;  
046 }  
047  
048 /\*\*  
049 \* Gets the iterator being decorated.  
050 \*  
051 \* @return the decorated iterator  
052 \*/  
053 protected ListIterator<E> getListIterator() {  
054 return iterator;  
055 }  
056  
057 //-----------------------------------------------------------------------  
058  
059 /\*\* {@inheritDoc} \*/  
060 @Override  
061 public boolean hasNext() {  
062 return iterator.hasNext();  
063 }  
064  
065 /\*\* {@inheritDoc} \*/  
066 @Override  
067 public E next() {  
068 return iterator.next();  
069 }  
070  
071 /\*\* {@inheritDoc} \*/  
072 @Override  
073 public int nextIndex() {  
074 return iterator.nextIndex();  
075 }  
076  
077 /\*\* {@inheritDoc} \*/  
078 @Override  
079 public boolean hasPrevious() {  
080 return iterator.hasPrevious();  
081 }  
082  
083 /\*\* {@inheritDoc} \*/  
084 @Override  
085 public E previous() {  
086 return iterator.previous();  
087 }  
088  
089 /\*\* {@inheritDoc} \*/  
090 @Override  
091 public int previousIndex() {  
092 return iterator.previousIndex();  
093 }  
094  
095 /\*\* {@inheritDoc} \*/  
096 @Override  
097 public void remove() {  
098 iterator.remove();  
099 }  
100  
101 /\*\* {@inheritDoc} \*/  
102 @Override  
103 public void set(final E obj) {  
104 iterator.set(obj);  
105 }  
106  
107 /\*\* {@inheritDoc} \*/  
108 @Override  
109 public void add(final E obj) {  
110 iterator.add(obj);  
111 }  
112  
113}