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 org.apache.commons.collections4.OrderedMapIterator;  
020  
021/\*\*  
022 \* Provides basic behaviour for decorating an ordered map iterator with extra functionality.  
023 \* <p>  
024 \* All methods are forwarded to the decorated map iterator.  
025 \*  
026 \* @param <K> the type of keys  
027 \* @param <V> the type of mapped values  
028 \* @since 3.0  
029 \*/  
030public class AbstractOrderedMapIteratorDecorator<K, V> implements OrderedMapIterator<K, V> {  
031  
032 /\*\* The iterator being decorated \*/  
033 private final OrderedMapIterator<K, V> iterator;  
034  
035 //-----------------------------------------------------------------------  
036 /\*\*  
037 \* Constructor that decorates the specified iterator.  
038 \*  
039 \* @param iterator the iterator to decorate, must not be null  
040 \* @throws NullPointerException if the iterator is null  
041 \*/  
042 public AbstractOrderedMapIteratorDecorator(final OrderedMapIterator<K, V> iterator) {  
043 super();  
044 if (iterator == null) {  
045 throw new NullPointerException("OrderedMapIterator must not be null");  
046 }  
047 this.iterator = iterator;  
048 }  
049  
050 /\*\*  
051 \* Gets the iterator being decorated.  
052 \*  
053 \* @return the decorated iterator  
054 \*/  
055 protected OrderedMapIterator<K, V> getOrderedMapIterator() {  
056 return iterator;  
057 }  
058  
059 //-----------------------------------------------------------------------  
060  
061 /\*\* {@inheritDoc} \*/  
062 @Override  
063 public boolean hasNext() {  
064 return iterator.hasNext();  
065 }  
066  
067 /\*\* {@inheritDoc} \*/  
068 @Override  
069 public K next() {  
070 return iterator.next();  
071 }  
072  
073 /\*\* {@inheritDoc} \*/  
074 @Override  
075 public boolean hasPrevious() {  
076 return iterator.hasPrevious();  
077 }  
078  
079 /\*\* {@inheritDoc} \*/  
080 @Override  
081 public K previous() {  
082 return iterator.previous();  
083 }  
084  
085 /\*\* {@inheritDoc} \*/  
086 @Override  
087 public void remove() {  
088 iterator.remove();  
089 }  
090  
091 /\*\* {@inheritDoc} \*/  
092 @Override  
093 public K getKey() {  
094 return iterator.getKey();  
095 }  
096  
097 /\*\* {@inheritDoc} \*/  
098 @Override  
099 public V getValue() {  
100 return iterator.getValue();  
101 }  
102  
103 /\*\* {@inheritDoc} \*/  
104 @Override  
105 public V setValue(final V obj) {  
106 return iterator.setValue(obj);  
107 }  
108  
109}