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.map;  
018  
019import java.util.Iterator;  
020import java.util.Map;  
021import java.util.Set;  
022  
023import org.apache.commons.collections4.MapIterator;  
024import org.apache.commons.collections4.ResettableIterator;  
025  
026/\*\*  
027 \* Adapts a Map entrySet to the MapIterator interface.  
028 \*  
029 \* @param <K> the type of the keys in the map  
030 \* @param <V> the type of the values in the map  
031 \*  
032 \* @since 4.0  
033 \*/  
034public class EntrySetToMapIteratorAdapter<K, V> implements MapIterator<K, V>, ResettableIterator<K> {  
035  
036 /\*\* The adapted Map entry Set. \*/  
037 Set<Map.Entry<K, V>> entrySet;  
038  
039 /\*\* The resettable iterator in use. \*/  
040 transient Iterator<Map.Entry<K, V>> iterator;  
041  
042 /\*\* The currently positioned Map entry. \*/  
043 transient Map.Entry<K, V> entry;  
044  
045 /\*\*  
046 \* Create a new EntrySetToMapIteratorAdapter.  
047 \* @param entrySet the entrySet to adapt  
048 \*/  
049 public EntrySetToMapIteratorAdapter(final Set<Map.Entry<K, V>> entrySet) {  
050 this.entrySet = entrySet;  
051 reset();  
052 }  
053  
054 /\*\*  
055 \* {@inheritDoc}  
056 \*/  
057 @Override  
058 public K getKey() {  
059 return current().getKey();  
060 }  
061  
062 /\*\*  
063 \* {@inheritDoc}  
064 \*/  
065 @Override  
066 public V getValue() {  
067 return current().getValue();  
068 }  
069  
070 /\*\*  
071 \* {@inheritDoc}  
072 \*/  
073 @Override  
074 public V setValue(final V value) {  
075 return current().setValue(value);  
076 }  
077  
078 /\*\*  
079 \* {@inheritDoc}  
080 \*/  
081 @Override  
082 public boolean hasNext() {  
083 return iterator.hasNext();  
084 }  
085  
086 /\*\*  
087 \* {@inheritDoc}  
088 \*/  
089 @Override  
090 public K next() {  
091 entry = iterator.next();  
092 return getKey();  
093 }  
094  
095 /\*\*  
096 \* {@inheritDoc}  
097 \*/  
098 @Override  
099 public synchronized void reset() {  
100 iterator = entrySet.iterator();  
101 }  
102  
103 /\*\*  
104 \* {@inheritDoc}  
105 \*/  
106 @Override  
107 public void remove() {  
108 iterator.remove();  
109 entry = null;  
110 }  
111  
112 /\*\*  
113 \* Get the currently active entry.  
114 \* @return Map.Entry<K, V>  
115 \*/  
116 protected synchronized Map.Entry<K, V> current() {  
117 if (entry == null) {  
118 throw new IllegalStateException();  
119 }  
120 return entry;  
121 }  
122}