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;  
018  
019import java.util.Iterator;  
020  
021/\*\*  
022 \* Defines an iterator that operates over a <code>Map</code>.  
023 \* <p>  
024 \* This iterator is a special version designed for maps. It can be more  
025 \* efficient to use this rather than an entry set iterator where the option  
026 \* is available, and it is certainly more convenient.  
027 \* </p>  
028 \* <p>  
029 \* A map that provides this interface may not hold the data internally using  
030 \* Map Entry objects, thus this interface can avoid lots of object creation.  
031 \* </p>  
032 \* <p>  
033 \* In use, this iterator iterates through the keys in the map. After each call  
034 \* to <code>next()</code>, the <code>getValue()</code> method provides direct  
035 \* access to the value. The value can also be set using <code>setValue()</code>.  
036 \* </p>  
037 \* <pre>{@code  
038 \* MapIterator<String,Integer> it = map.mapIterator();  
039 \* while (it.hasNext()) {  
040 \* String key = it.next();  
041 \* Integer value = it.getValue();  
042 \* it.setValue(value + 1);  
043 \* }  
044 \* }</pre>  
045 \*  
046 \* @param <K> the type of the keys in the map  
047 \* @param <V> the type of the values in the map  
048 \* @since 3.0  
049 \*/  
050public interface MapIterator<K, V> extends Iterator<K> {  
051  
052 /\*\*  
053 \* Checks to see if there are more entries still to be iterated.  
054 \*  
055 \* @return <code>true</code> if the iterator has more elements  
056 \*/  
057 @Override  
058 boolean hasNext();  
059  
060 /\*\*  
061 \* Gets the next <em>key</em> from the <code>Map</code>.  
062 \*  
063 \* @return the next key in the iteration  
064 \* @throws java.util.NoSuchElementException if the iteration is finished  
065 \*/  
066 @Override  
067 K next();  
068  
069 //-----------------------------------------------------------------------  
070 /\*\*  
071 \* Gets the current key, which is the key returned by the last call  
072 \* to <code>next()</code>.  
073 \*  
074 \* @return the current key  
075 \* @throws IllegalStateException if <code>next()</code> has not yet been called  
076 \*/  
077 K getKey();  
078  
079 /\*\*  
080 \* Gets the current value, which is the value associated with the last key  
081 \* returned by <code>next()</code>.  
082 \*  
083 \* @return the current value  
084 \* @throws IllegalStateException if <code>next()</code> has not yet been called  
085 \*/  
086 V getValue();  
087  
088 //-----------------------------------------------------------------------  
089 /\*\*  
090 \* Removes the last returned key from the underlying <code>Map</code> (optional operation).  
091 \* <p>  
092 \* This method can be called once per call to <code>next()</code>.  
093 \*  
094 \* @throws UnsupportedOperationException if remove is not supported by the map  
095 \* @throws IllegalStateException if <code>next()</code> has not yet been called  
096 \* @throws IllegalStateException if <code>remove()</code> has already been called  
097 \* since the last call to <code>next()</code>  
098 \*/  
099 @Override  
100 void remove();  
101  
102 /\*\*  
103 \* Sets the value associated with the current key (optional operation).  
104 \*  
105 \* @param value the new value  
106 \* @return the previous value  
107 \* @throws UnsupportedOperationException if setValue is not supported by the map  
108 \* @throws IllegalStateException if <code>next()</code> has not yet been called  
109 \* @throws IllegalStateException if <code>remove()</code> has been called since the  
110 \* last call to <code>next()</code>  
111 \*/  
112 V setValue(V value);  
113  
114}