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.multimap;  
018  
019import java.util.Collection;  
020import java.util.Map;  
021import java.util.Map.Entry;  
022import java.util.Set;  
023  
024import org.apache.commons.collections4.MapIterator;  
025import org.apache.commons.collections4.MultiSet;  
026import org.apache.commons.collections4.MultiValuedMap;  
027import org.apache.commons.collections4.Unmodifiable;  
028import org.apache.commons.collections4.collection.UnmodifiableCollection;  
029import org.apache.commons.collections4.iterators.UnmodifiableMapIterator;  
030import org.apache.commons.collections4.map.UnmodifiableMap;  
031import org.apache.commons.collections4.multiset.UnmodifiableMultiSet;  
032import org.apache.commons.collections4.set.UnmodifiableSet;  
033  
034/\*\*  
035 \* Decorates another {@link MultiValuedMap} to ensure it can't be altered.  
036 \* <p>  
037 \* Attempts to modify it will result in an UnsupportedOperationException.  
038 \* </p>  
039 \*  
040 \* @param <K> the type of key elements  
041 \* @param <V> the type of value elements  
042 \*  
043 \* @since 4.1  
044 \*/  
045public final class UnmodifiableMultiValuedMap<K, V>  
046 extends AbstractMultiValuedMapDecorator<K, V> implements Unmodifiable {  
047  
048 /\*\* Serialization version \*/  
049 private static final long serialVersionUID = 20150612L;  
050  
051 /\*\*  
052 \* Factory method to create an unmodifiable MultiValuedMap.  
053 \* <p>  
054 \* If the map passed in is already unmodifiable, it is returned.  
055 \*  
056 \* @param <K> the type of key elements  
057 \* @param <V> the type of value elements  
058 \* @param map the map to decorate, may not be null  
059 \* @return an unmodifiable MultiValuedMap  
060 \* @throws NullPointerException if map is null  
061 \*/  
062 @SuppressWarnings("unchecked")  
063 public static <K, V> UnmodifiableMultiValuedMap<K, V> unmodifiableMultiValuedMap(  
064 final MultiValuedMap<? extends K, ? extends V> map) {  
065 if (map instanceof Unmodifiable) {  
066 return (UnmodifiableMultiValuedMap<K, V>) map;  
067 }  
068 return new UnmodifiableMultiValuedMap<>(map);  
069 }  
070  
071 /\*\*  
072 \* Constructor that wraps (not copies).  
073 \*  
074 \* @param map the MultiValuedMap to decorate, may not be null  
075 \* @throws NullPointerException if the map is null  
076 \*/  
077 @SuppressWarnings("unchecked")  
078 private UnmodifiableMultiValuedMap(final MultiValuedMap<? extends K, ? extends V> map) {  
079 super((MultiValuedMap<K, V>) map);  
080 }  
081  
082 @Override  
083 public Collection<V> remove(final Object key) {  
084 throw new UnsupportedOperationException();  
085 }  
086  
087 @Override  
088 public boolean removeMapping(final Object key, final Object item) {  
089 throw new UnsupportedOperationException();  
090 }  
091  
092 @Override  
093 public void clear() {  
094 throw new UnsupportedOperationException();  
095 }  
096  
097 @Override  
098 public Collection<V> get(final K key) {  
099 return UnmodifiableCollection.unmodifiableCollection(decorated().get(key));  
100 }  
101  
102 @Override  
103 public boolean put(final K key, final V value) {  
104 throw new UnsupportedOperationException();  
105 }  
106  
107 @Override  
108 public Set<K> keySet() {  
109 return UnmodifiableSet.unmodifiableSet(decorated().keySet());  
110 }  
111  
112 @Override  
113 public Collection<Entry<K, V>> entries() {  
114 return UnmodifiableCollection.unmodifiableCollection(decorated().entries());  
115 }  
116  
117 @Override  
118 public MultiSet<K> keys() {  
119 return UnmodifiableMultiSet.unmodifiableMultiSet(decorated().keys());  
120 }  
121  
122 @Override  
123 public Collection<V> values() {  
124 return UnmodifiableCollection.unmodifiableCollection(decorated().values());  
125 }  
126  
127 @Override  
128 public Map<K, Collection<V>> asMap() {  
129 return UnmodifiableMap.unmodifiableMap(decorated().asMap());  
130 }  
131  
132 @Override  
133 public MapIterator<K, V> mapIterator() {  
134 return UnmodifiableMapIterator.unmodifiableMapIterator(decorated().mapIterator());  
135 }  
136  
137 @Override  
138 public boolean putAll(final K key, final Iterable<? extends V> values) {  
139 throw new UnsupportedOperationException();  
140 }  
141  
142 @Override  
143 public boolean putAll(final Map<? extends K, ? extends V> map) {  
144 throw new UnsupportedOperationException();  
145 }  
146  
147 @Override  
148 public boolean putAll(final MultiValuedMap<? extends K, ? extends V> map) {  
149 throw new UnsupportedOperationException();  
150 }  
151  
152}