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.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.io.Serializable;  
023import java.util.Collection;  
024import java.util.Map;  
025import java.util.Set;  
026  
027import org.apache.commons.collections4.IterableMap;  
028import org.apache.commons.collections4.MapIterator;  
029import org.apache.commons.collections4.Unmodifiable;  
030import org.apache.commons.collections4.collection.UnmodifiableCollection;  
031import org.apache.commons.collections4.iterators.EntrySetMapIterator;  
032import org.apache.commons.collections4.iterators.UnmodifiableMapIterator;  
033import org.apache.commons.collections4.set.UnmodifiableSet;  
034  
035/\*\*  
036 \* Decorates another <code>Map</code> to ensure it can't be altered.  
037 \* <p>  
038 \* This class is Serializable from Commons Collections 3.1.  
039 \* </p>  
040 \* <p>  
041 \* Attempts to modify it will result in an UnsupportedOperationException.  
042 \* </p>  
043 \*  
044 \* @param <K> the type of the keys in this map  
045 \* @param <V> the type of the values in this map  
046 \* @since 3.0  
047 \*/  
048public final class UnmodifiableMap<K, V>  
049 extends AbstractMapDecorator<K, V>  
050 implements Unmodifiable, Serializable {  
051  
052 /\*\* Serialization version \*/  
053 private static final long serialVersionUID = 2737023427269031941L;  
054  
055 /\*\*  
056 \* Factory method to create an unmodifiable map.  
057 \*  
058 \* @param <K> the key type  
059 \* @param <V> the value type  
060 \* @param map the map to decorate, must not be null  
061 \* @return a new unmodifiable map  
062 \* @throws NullPointerException if map is null  
063 \* @since 4.0  
064 \*/  
065 public static <K, V> Map<K, V> unmodifiableMap(final Map<? extends K, ? extends V> map) {  
066 if (map instanceof Unmodifiable) {  
067 @SuppressWarnings("unchecked") // safe to upcast  
068 final Map<K, V> tmpMap = (Map<K, V>) map;  
069 return tmpMap;  
070 }  
071 return new UnmodifiableMap<>(map);  
072 }  
073  
074 //-----------------------------------------------------------------------  
075 /\*\*  
076 \* Constructor that wraps (not copies).  
077 \*  
078 \* @param map the map to decorate, must not be null  
079 \* @throws NullPointerException if map is null  
080 \*/  
081 @SuppressWarnings("unchecked") // safe to upcast  
082 private UnmodifiableMap(final Map<? extends K, ? extends V> map) {  
083 super((Map<K, V>) map);  
084 }  
085  
086 //-----------------------------------------------------------------------  
087 /\*\*  
088 \* Write the map out using a custom routine.  
089 \*  
090 \* @param out the output stream  
091 \* @throws IOException if an error occurs while writing to the stream  
092 \* @since 3.1  
093 \*/  
094 private void writeObject(final ObjectOutputStream out) throws IOException {  
095 out.defaultWriteObject();  
096 out.writeObject(map);  
097 }  
098  
099 /\*\*  
100 \* Read the map in using a custom routine.  
101 \*  
102 \* @param in the input stream  
103 \* @throws IOException if an error occurs while reading from the stream  
104 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
105 \* @since 3.1  
106 \*/  
107 @SuppressWarnings("unchecked")  
108 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
109 in.defaultReadObject();  
110 map = (Map<K, V>) in.readObject();  
111 }  
112  
113 //-----------------------------------------------------------------------  
114 @Override  
115 public void clear() {  
116 throw new UnsupportedOperationException();  
117 }  
118  
119 @Override  
120 public V put(final K key, final V value) {  
121 throw new UnsupportedOperationException();  
122 }  
123  
124 @Override  
125 public void putAll(final Map<? extends K, ? extends V> mapToCopy) {  
126 throw new UnsupportedOperationException();  
127 }  
128  
129 @Override  
130 public V remove(final Object key) {  
131 throw new UnsupportedOperationException();  
132 }  
133  
134 @Override  
135 public MapIterator<K, V> mapIterator() {  
136 if (map instanceof IterableMap) {  
137 final MapIterator<K, V> it = ((IterableMap<K, V>) map).mapIterator();  
138 return UnmodifiableMapIterator.unmodifiableMapIterator(it);  
139 }  
140 final MapIterator<K, V> it = new EntrySetMapIterator<>(map);  
141 return UnmodifiableMapIterator.unmodifiableMapIterator(it);  
142 }  
143  
144 @Override  
145 public Set<Map.Entry<K, V>> entrySet() {  
146 final Set<Map.Entry<K, V>> set = super.entrySet();  
147 return UnmodifiableEntrySet.unmodifiableEntrySet(set);  
148 }  
149  
150 @Override  
151 public Set<K> keySet() {  
152 final Set<K> set = super.keySet();  
153 return UnmodifiableSet.unmodifiableSet(set);  
154 }  
155  
156 @Override  
157 public Collection<V> values() {  
158 final Collection<V> coll = super.values();  
159 return UnmodifiableCollection.unmodifiableCollection(coll);  
160 }  
161  
162}