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.OrderedMap;  
028import org.apache.commons.collections4.OrderedMapIterator;  
029import org.apache.commons.collections4.Unmodifiable;  
030import org.apache.commons.collections4.collection.UnmodifiableCollection;  
031import org.apache.commons.collections4.iterators.UnmodifiableOrderedMapIterator;  
032import org.apache.commons.collections4.set.UnmodifiableSet;  
033  
034/\*\*  
035 \* Decorates another <code>OrderedMap</code> to ensure it can't be altered.  
036 \* <p>  
037 \* This class is Serializable from Commons Collections 3.1.  
038 \* </p>  
039 \* <p>  
040 \* Attempts to modify it will result in an UnsupportedOperationException.  
041 \* </p>  
042 \*  
043 \* @param <K> the type of the keys in this map  
044 \* @param <V> the type of the values in this map  
045 \* @since 3.0  
046 \*/  
047public final class UnmodifiableOrderedMap<K, V> extends AbstractOrderedMapDecorator<K, V> implements  
048 Unmodifiable, Serializable {  
049  
050 /\*\* Serialization version \*/  
051 private static final long serialVersionUID = 8136428161720526266L;  
052  
053 /\*\*  
054 \* Factory method to create an unmodifiable sorted map.  
055 \*  
056 \* @param <K> the key type  
057 \* @param <V> the value type  
058 \* @param map the map to decorate, must not be null  
059 \* @return a new ordered map  
060 \* @throws NullPointerException if map is null  
061 \* @since 4.0  
062 \*/  
063 public static <K, V> OrderedMap<K, V> unmodifiableOrderedMap(final OrderedMap<? extends K, ? extends V> map) {  
064 if (map instanceof Unmodifiable) {  
065 @SuppressWarnings("unchecked") // safe to upcast  
066 final OrderedMap<K, V> tmpMap = (OrderedMap<K, V>) map;  
067 return tmpMap;  
068 }  
069 return new UnmodifiableOrderedMap<>(map);  
070 }  
071  
072 //-----------------------------------------------------------------------  
073 /\*\*  
074 \* Constructor that wraps (not copies).  
075 \*  
076 \* @param map the map to decorate, must not be null  
077 \* @throws NullPointerException if map is null  
078 \*/  
079 @SuppressWarnings("unchecked") // safe to upcast  
080 private UnmodifiableOrderedMap(final OrderedMap<? extends K, ? extends V> map) {  
081 super((OrderedMap<K, V>) map);  
082 }  
083  
084 //-----------------------------------------------------------------------  
085 /\*\*  
086 \* Write the map out using a custom routine.  
087 \*  
088 \* @param out the output stream  
089 \* @throws IOException if an error occurs while writing to the stream  
090 \* @since 3.1  
091 \*/  
092 private void writeObject(final ObjectOutputStream out) throws IOException {  
093 out.defaultWriteObject();  
094 out.writeObject(map);  
095 }  
096  
097 /\*\*  
098 \* Read the map in using a custom routine.  
099 \*  
100 \* @param in the input stream  
101 \* @throws IOException if an error occurs while reading from the stream  
102 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
103 \* @since 3.1  
104 \*/  
105 @SuppressWarnings("unchecked") // (1) should only fail if input stream is incorrect  
106 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
107 in.defaultReadObject();  
108 map = (Map<K, V>) in.readObject(); // (1)  
109 }  
110  
111 //-----------------------------------------------------------------------  
112 @Override  
113 public OrderedMapIterator<K, V> mapIterator() {  
114 final OrderedMapIterator<K, V> it = decorated().mapIterator();  
115 return UnmodifiableOrderedMapIterator.unmodifiableOrderedMapIterator(it);  
116 }  
117  
118 @Override  
119 public void clear() {  
120 throw new UnsupportedOperationException();  
121 }  
122  
123 @Override  
124 public V put(final K key, final V value) {  
125 throw new UnsupportedOperationException();  
126 }  
127  
128 @Override  
129 public void putAll(final Map<? extends K, ? extends V> mapToCopy) {  
130 throw new UnsupportedOperationException();  
131 }  
132  
133 @Override  
134 public V remove(final Object key) {  
135 throw new UnsupportedOperationException();  
136 }  
137  
138 @Override  
139 public Set<Map.Entry<K, V>> entrySet() {  
140 final Set<Map.Entry<K, V>> set = super.entrySet();  
141 return UnmodifiableEntrySet.unmodifiableEntrySet(set);  
142 }  
143  
144 @Override  
145 public Set<K> keySet() {  
146 final Set<K> set = super.keySet();  
147 return UnmodifiableSet.unmodifiableSet(set);  
148 }  
149  
150 @Override  
151 public Collection<V> values() {  
152 final Collection<V> coll = super.values();  
153 return UnmodifiableCollection.unmodifiableCollection(coll);  
154 }  
155  
156}