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.Comparator;  
025import java.util.Map;  
026import java.util.Set;  
027import java.util.SortedMap;  
028  
029import org.apache.commons.collections4.Unmodifiable;  
030import org.apache.commons.collections4.collection.UnmodifiableCollection;  
031import org.apache.commons.collections4.set.UnmodifiableSet;  
032  
033/\*\*  
034 \* Decorates another <code>SortedMap</code> to ensure it can't be altered.  
035 \* <p>  
036 \* This class is Serializable from Commons Collections 3.1.  
037 \* </p>  
038 \* <p>  
039 \* Attempts to modify it will result in an UnsupportedOperationException.  
040 \* </p>  
041 \*  
042 \* @param <K> the type of the keys in this map  
043 \* @param <V> the type of the values in this map  
044 \* @since 3.0  
045 \*/  
046public final class UnmodifiableSortedMap<K, V>  
047 extends AbstractSortedMapDecorator<K, V>  
048 implements Unmodifiable, Serializable {  
049  
050 /\*\* Serialization version \*/  
051 private static final long serialVersionUID = 5805344239827376360L;  
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 unmodifiable sorted map  
060 \* @throws NullPointerException if map is null  
061 \* @since 4.0  
062 \*/  
063 public static <K, V> SortedMap<K, V> unmodifiableSortedMap(final SortedMap<K, ? extends V> map) {  
064 if (map instanceof Unmodifiable) {  
065 @SuppressWarnings("unchecked") // safe to upcast  
066 final SortedMap<K, V> tmpMap = (SortedMap<K, V>) map;  
067 return tmpMap;  
068 }  
069 return new UnmodifiableSortedMap<>(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 UnmodifiableSortedMap(final SortedMap<K, ? extends V> map) {  
081 super((SortedMap<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")  
106 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
107 in.defaultReadObject();  
108 map = (Map<K, V>) in.readObject();  
109 }  
110  
111 //-----------------------------------------------------------------------  
112 @Override  
113 public void clear() {  
114 throw new UnsupportedOperationException();  
115 }  
116  
117 @Override  
118 public V put(final K key, final V value) {  
119 throw new UnsupportedOperationException();  
120 }  
121  
122 @Override  
123 public void putAll(final Map<? extends K, ? extends V> mapToCopy) {  
124 throw new UnsupportedOperationException();  
125 }  
126  
127 @Override  
128 public V remove(final Object key) {  
129 throw new UnsupportedOperationException();  
130 }  
131  
132 @Override  
133 public Set<Map.Entry<K, V>> entrySet() {  
134 return UnmodifiableEntrySet.unmodifiableEntrySet(super.entrySet());  
135 }  
136  
137 @Override  
138 public Set<K> keySet() {  
139 return UnmodifiableSet.unmodifiableSet(super.keySet());  
140 }  
141  
142 @Override  
143 public Collection<V> values() {  
144 return UnmodifiableCollection.unmodifiableCollection(super.values());  
145 }  
146  
147 //-----------------------------------------------------------------------  
148 @Override  
149 public K firstKey() {  
150 return decorated().firstKey();  
151 }  
152  
153 @Override  
154 public K lastKey() {  
155 return decorated().lastKey();  
156 }  
157  
158 @Override  
159 public Comparator<? super K> comparator() {  
160 return decorated().comparator();  
161 }  
162  
163 @Override  
164 public SortedMap<K, V> subMap(final K fromKey, final K toKey) {  
165 return new UnmodifiableSortedMap<>(decorated().subMap(fromKey, toKey));  
166 }  
167  
168 @Override  
169 public SortedMap<K, V> headMap(final K toKey) {  
170 return new UnmodifiableSortedMap<>(decorated().headMap(toKey));  
171 }  
172  
173 @Override  
174 public SortedMap<K, V> tailMap(final K fromKey) {  
175 return new UnmodifiableSortedMap<>(decorated().tailMap(fromKey));  
176 }  
177  
178}