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.multiset;  
018  
019import java.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.util.Collection;  
023import java.util.Iterator;  
024import java.util.Set;  
025import java.util.function.Predicate;  
026  
027import org.apache.commons.collections4.MultiSet;  
028import org.apache.commons.collections4.Unmodifiable;  
029import org.apache.commons.collections4.iterators.UnmodifiableIterator;  
030import org.apache.commons.collections4.set.UnmodifiableSet;  
031  
032/\*\*  
033 \* Decorates another {@link MultiSet} to ensure it can't be altered.  
034 \* <p>  
035 \* Attempts to modify it will result in an UnsupportedOperationException.  
036 \* </p>  
037 \*  
038 \* @param <E> the type held in the multiset  
039 \* @since 4.1  
040 \*/  
041public final class UnmodifiableMultiSet<E>  
042 extends AbstractMultiSetDecorator<E> implements Unmodifiable {  
043  
044 /\*\* Serialization version \*/  
045 private static final long serialVersionUID = 20150611L;  
046  
047 /\*\*  
048 \* Factory method to create an unmodifiable multiset.  
049 \* <p>  
050 \* If the multiset passed in is already unmodifiable, it is returned.  
051 \*  
052 \* @param <E> the type of the elements in the multiset  
053 \* @param multiset the multiset to decorate, may not be null  
054 \* @return an unmodifiable MultiSet  
055 \* @throws NullPointerException if multiset is null  
056 \*/  
057 public static <E> MultiSet<E> unmodifiableMultiSet(final MultiSet<? extends E> multiset) {  
058 if (multiset instanceof Unmodifiable) {  
059 @SuppressWarnings("unchecked") // safe to upcast  
060 final MultiSet<E> tmpMultiSet = (MultiSet<E>) multiset;  
061 return tmpMultiSet;  
062 }  
063 return new UnmodifiableMultiSet<>(multiset);  
064 }  
065  
066 //-----------------------------------------------------------------------  
067 /\*\*  
068 \* Constructor that wraps (not copies).  
069 \*  
070 \* @param multiset the multiset to decorate, may not be null  
071 \* @throws NullPointerException if multiset is null  
072 \*/  
073 @SuppressWarnings("unchecked") // safe to upcast  
074 private UnmodifiableMultiSet(final MultiSet<? extends E> multiset) {  
075 super((MultiSet<E>) multiset);  
076 }  
077  
078 //-----------------------------------------------------------------------  
079 /\*\*  
080 \* Write the collection out using a custom routine.  
081 \*  
082 \* @param out the output stream  
083 \* @throws IOException if an error occurs while writing to the stream  
084 \*/  
085 private void writeObject(final ObjectOutputStream out) throws IOException {  
086 out.defaultWriteObject();  
087 out.writeObject(decorated());  
088 }  
089  
090 /\*\*  
091 \* Read the collection in using a custom routine.  
092 \*  
093 \* @param in the input stream  
094 \* @throws IOException if an error occurs while reading from the stream  
095 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
096 \* @throws ClassCastException if deserialised object has wrong type  
097 \*/  
098 @SuppressWarnings("unchecked") // will throw CCE, see Javadoc  
099 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
100 in.defaultReadObject();  
101 setCollection((Collection<E>) in.readObject());  
102 }  
103  
104 //-----------------------------------------------------------------------  
105 @Override  
106 public Iterator<E> iterator() {  
107 return UnmodifiableIterator.<E> unmodifiableIterator(decorated().iterator());  
108 }  
109  
110 @Override  
111 public boolean add(final E object) {  
112 throw new UnsupportedOperationException();  
113 }  
114  
115 @Override  
116 public boolean addAll(final Collection<? extends E> coll) {  
117 throw new UnsupportedOperationException();  
118 }  
119  
120 @Override  
121 public void clear() {  
122 throw new UnsupportedOperationException();  
123 }  
124  
125 @Override  
126 public boolean remove(final Object object) {  
127 throw new UnsupportedOperationException();  
128 }  
129  
130 /\*\*  
131 \* @since 4.4  
132 \*/  
133 @Override  
134 public boolean removeIf(Predicate<? super E> filter) {  
135 throw new UnsupportedOperationException();  
136 }  
137  
138 @Override  
139 public boolean removeAll(final Collection<?> coll) {  
140 throw new UnsupportedOperationException();  
141 }  
142  
143 @Override  
144 public boolean retainAll(final Collection<?> coll) {  
145 throw new UnsupportedOperationException();  
146 }  
147  
148 //-----------------------------------------------------------------------  
149 @Override  
150 public int setCount(final E object, final int count) {  
151 throw new UnsupportedOperationException();  
152 }  
153  
154 @Override  
155 public int add(final E object, final int count) {  
156 throw new UnsupportedOperationException();  
157 }  
158  
159 @Override  
160 public int remove(final Object object, final int count) {  
161 throw new UnsupportedOperationException();  
162 }  
163  
164 @Override  
165 public Set<E> uniqueSet() {  
166 final Set<E> set = decorated().uniqueSet();  
167 return UnmodifiableSet.unmodifiableSet(set);  
168 }  
169  
170 @Override  
171 public Set<MultiSet.Entry<E>> entrySet() {  
172 final Set<MultiSet.Entry<E>> set = decorated().entrySet();  
173 return UnmodifiableSet.unmodifiableSet(set);  
174 }  
175  
176}