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.bag;  
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.SortedBag;  
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 SortedBag} to ensure it can't be altered.  
034 \* <p>  
035 \* This class is Serializable from Commons Collections 3.1.  
036 \* </p>  
037 \* <p>  
038 \* Attempts to modify it will result in an UnsupportedOperationException.  
039 \* </p>  
040 \*  
041 \* @param <E> the type of elements in this bag  
042 \* @since 3.0  
043 \*/  
044public final class UnmodifiableSortedBag<E>  
045 extends AbstractSortedBagDecorator<E> implements Unmodifiable {  
046  
047 /\*\* Serialization version \*/  
048 private static final long serialVersionUID = -3190437252665717841L;  
049  
050 /\*\*  
051 \* Factory method to create an unmodifiable bag.  
052 \* <p>  
053 \* If the bag passed in is already unmodifiable, it is returned.  
054 \*  
055 \* @param <E> the type of the elements in the bag  
056 \* @param bag the bag to decorate, must not be null  
057 \* @return an unmodifiable SortedBag  
058 \* @throws NullPointerException if bag is null  
059 \* @since 4.0  
060 \*/  
061 public static <E> SortedBag<E> unmodifiableSortedBag(final SortedBag<E> bag) {  
062 if (bag instanceof Unmodifiable) {  
063 return bag;  
064 }  
065 return new UnmodifiableSortedBag<>(bag);  
066 }  
067  
068 //-----------------------------------------------------------------------  
069 /\*\*  
070 \* Constructor that wraps (not copies).  
071 \*  
072 \* @param bag the bag to decorate, must not be null  
073 \* @throws NullPointerException if bag is null  
074 \*/  
075 private UnmodifiableSortedBag(final SortedBag<E> bag) {  
076 super(bag);  
077 }  
078  
079 //-----------------------------------------------------------------------  
080 /\*\*  
081 \* Write the collection out using a custom routine.  
082 \*  
083 \* @param out the output stream  
084 \* @throws IOException if an error occurs while writing to the stream  
085 \*/  
086 private void writeObject(final ObjectOutputStream out) throws IOException {  
087 out.defaultWriteObject();  
088 out.writeObject(decorated());  
089 }  
090  
091 /\*\*  
092 \* Read the collection in using a custom routine.  
093 \*  
094 \* @param in the input stream  
095 \* @throws IOException if an error occurs while reading from the stream  
096 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
097 \* @throws ClassCastException if deserialised object has wrong type  
098 \*/  
099 @SuppressWarnings("unchecked") // will throw CCE, see Javadoc  
100 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
101 in.defaultReadObject();  
102 setCollection((Collection<E>) in.readObject());  
103 }  
104  
105 //-----------------------------------------------------------------------  
106 @Override  
107 public Iterator<E> iterator() {  
108 return UnmodifiableIterator.unmodifiableIterator(decorated().iterator());  
109 }  
110  
111 @Override  
112 public boolean add(final E object) {  
113 throw new UnsupportedOperationException();  
114 }  
115  
116 @Override  
117 public boolean addAll(final Collection<? extends E> coll) {  
118 throw new UnsupportedOperationException();  
119 }  
120  
121 @Override  
122 public void clear() {  
123 throw new UnsupportedOperationException();  
124 }  
125  
126 @Override  
127 public boolean remove(final Object object) {  
128 throw new UnsupportedOperationException();  
129 }  
130  
131 /\*\*  
132 \* @since 4.4  
133 \*/  
134 @Override  
135 public boolean removeIf(Predicate<? super E> filter) {  
136 throw new UnsupportedOperationException();  
137 }  
138  
139 @Override  
140 public boolean removeAll(final Collection<?> coll) {  
141 throw new UnsupportedOperationException();  
142 }  
143  
144 @Override  
145 public boolean retainAll(final Collection<?> coll) {  
146 throw new UnsupportedOperationException();  
147 }  
148  
149 //-----------------------------------------------------------------------  
150 @Override  
151 public boolean add(final E object, final int count) {  
152 throw new UnsupportedOperationException();  
153 }  
154  
155 @Override  
156 public boolean remove(final Object object, final int count) {  
157 throw new UnsupportedOperationException();  
158 }  
159  
160 @Override  
161 public Set<E> uniqueSet() {  
162 final Set<E> set = decorated().uniqueSet();  
163 return UnmodifiableSet.unmodifiableSet(set);  
164 }  
165  
166}