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.Bag;  
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 Bag} 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 UnmodifiableBag<E>  
045 extends AbstractBagDecorator<E> implements Unmodifiable {  
046  
047 /\*\* Serialization version \*/  
048 private static final long serialVersionUID = -1873799975157099624L;  
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 Bag  
058 \* @throws NullPointerException if bag is null  
059 \* @since 4.0  
060 \*/  
061 public static <E> Bag<E> unmodifiableBag(final Bag<? extends E> bag) {  
062 if (bag instanceof Unmodifiable) {  
063 @SuppressWarnings("unchecked") // safe to upcast  
064 final Bag<E> tmpBag = (Bag<E>) bag;  
065 return tmpBag;  
066 }  
067 return new UnmodifiableBag<>(bag);  
068 }  
069  
070 //-----------------------------------------------------------------------  
071 /\*\*  
072 \* Constructor that wraps (not copies).  
073 \*  
074 \* @param bag the bag to decorate, must not be null  
075 \* @throws NullPointerException if bag is null  
076 \*/  
077 @SuppressWarnings("unchecked") // safe to upcast  
078 private UnmodifiableBag(final Bag<? extends E> bag) {  
079 super((Bag<E>) bag);  
080 }  
081  
082 //-----------------------------------------------------------------------  
083 /\*\*  
084 \* Write the collection out using a custom routine.  
085 \*  
086 \* @param out the output stream  
087 \* @throws IOException if an error occurs while writing to the stream  
088 \*/  
089 private void writeObject(final ObjectOutputStream out) throws IOException {  
090 out.defaultWriteObject();  
091 out.writeObject(decorated());  
092 }  
093  
094 /\*\*  
095 \* Read the collection in using a custom routine.  
096 \*  
097 \* @param in the input stream  
098 \* @throws IOException if an error occurs while reading from the stream  
099 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
100 \* @throws ClassCastException if deserialised object has wrong type  
101 \*/  
102 @SuppressWarnings("unchecked") // will throw CCE, see Javadoc  
103 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
104 in.defaultReadObject();  
105 setCollection((Collection<E>) in.readObject());  
106 }  
107  
108 //-----------------------------------------------------------------------  
109 @Override  
110 public Iterator<E> iterator() {  
111 return UnmodifiableIterator.<E> unmodifiableIterator(decorated().iterator());  
112 }  
113  
114 @Override  
115 public boolean add(final E object) {  
116 throw new UnsupportedOperationException();  
117 }  
118  
119 @Override  
120 public boolean addAll(final Collection<? extends E> coll) {  
121 throw new UnsupportedOperationException();  
122 }  
123  
124 @Override  
125 public void clear() {  
126 throw new UnsupportedOperationException();  
127 }  
128  
129 @Override  
130 public boolean remove(final Object object) {  
131 throw new UnsupportedOperationException();  
132 }  
133  
134 /\*\*  
135 \* @since 4.4  
136 \*/  
137 @Override  
138 public boolean removeIf(Predicate<? super E> filter) {  
139 throw new UnsupportedOperationException();  
140 }  
141  
142 @Override  
143 public boolean removeAll(final Collection<?> coll) {  
144 throw new UnsupportedOperationException();  
145 }  
146  
147 @Override  
148 public boolean retainAll(final Collection<?> coll) {  
149 throw new UnsupportedOperationException();  
150 }  
151  
152 //-----------------------------------------------------------------------  
153 @Override  
154 public boolean add(final E object, final int count) {  
155 throw new UnsupportedOperationException();  
156 }  
157  
158 @Override  
159 public boolean remove(final Object object, final int count) {  
160 throw new UnsupportedOperationException();  
161 }  
162  
163 @Override  
164 public Set<E> uniqueSet() {  
165 final Set<E> set = decorated().uniqueSet();  
166 return UnmodifiableSet.<E> unmodifiableSet(set);  
167 }  
168  
169}