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.queue;  
018  
019import java.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.util.Collection;  
023import java.util.Iterator;  
024import java.util.Queue;  
025import java.util.function.Predicate;  
026  
027import org.apache.commons.collections4.Unmodifiable;  
028import org.apache.commons.collections4.iterators.UnmodifiableIterator;  
029  
030/\*\*  
031 \* Decorates another {@link Queue} to ensure it can't be altered.  
032 \* <p>  
033 \* Attempts to modify it will result in an UnsupportedOperationException.  
034 \* </p>  
035 \*  
036 \* @param <E> the type of elements held in this queue  
037 \* @since 4.0  
038 \*/  
039public final class UnmodifiableQueue<E>  
040 extends AbstractQueueDecorator<E>  
041 implements Unmodifiable {  
042  
043 /\*\* Serialization version \*/  
044 private static final long serialVersionUID = 1832948656215393357L;  
045  
046 /\*\*  
047 \* Factory method to create an unmodifiable queue.  
048 \* <p>  
049 \* If the queue passed in is already unmodifiable, it is returned.  
050 \*  
051 \* @param <E> the type of the elements in the queue  
052 \* @param queue the queue to decorate, must not be null  
053 \* @return an unmodifiable Queue  
054 \* @throws NullPointerException if queue is null  
055 \*/  
056 public static <E> Queue<E> unmodifiableQueue(final Queue<? extends E> queue) {  
057 if (queue instanceof Unmodifiable) {  
058 @SuppressWarnings("unchecked") // safe to upcast  
059 final Queue<E> tmpQueue = (Queue<E>) queue;  
060 return tmpQueue;  
061 }  
062 return new UnmodifiableQueue<>(queue);  
063 }  
064  
065 //-----------------------------------------------------------------------  
066 /\*\*  
067 \* Constructor that wraps (not copies).  
068 \*  
069 \* @param queue the queue to decorate, must not be null  
070 \* @throws NullPointerException if queue is null  
071 \*/  
072 @SuppressWarnings("unchecked") // safe to upcast  
073 private UnmodifiableQueue(final Queue<? extends E> queue) {  
074 super((Queue<E>) queue);  
075 }  
076  
077 //-----------------------------------------------------------------------  
078 /\*\*  
079 \* Write the collection out using a custom routine.  
080 \*  
081 \* @param out the output stream  
082 \* @throws IOException if an I/O error occurs while writing to the output stream  
083 \*/  
084 private void writeObject(final ObjectOutputStream out) throws IOException {  
085 out.defaultWriteObject();  
086 out.writeObject(decorated());  
087 }  
088  
089 /\*\*  
090 \* Read the collection in using a custom routine.  
091 \*  
092 \* @param in the input stream  
093 \* @throws IOException if an I/O error occurs while reading from the input stream  
094 \* @throws ClassNotFoundException if the class of a serialized object can not be found  
095 \*/  
096 @SuppressWarnings("unchecked")  
097 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
098 in.defaultReadObject();  
099 setCollection((Collection<E>) in.readObject());  
100 }  
101  
102 //-----------------------------------------------------------------------  
103 @Override  
104 public Iterator<E> iterator() {  
105 return UnmodifiableIterator.unmodifiableIterator(decorated().iterator());  
106 }  
107  
108 @Override  
109 public boolean add(final Object object) {  
110 throw new UnsupportedOperationException();  
111 }  
112  
113 @Override  
114 public boolean addAll(final Collection<? extends E> coll) {  
115 throw new UnsupportedOperationException();  
116 }  
117  
118 @Override  
119 public void clear() {  
120 throw new UnsupportedOperationException();  
121 }  
122  
123 @Override  
124 public boolean remove(final Object object) {  
125 throw new UnsupportedOperationException();  
126 }  
127  
128 /\*\*  
129 \* @since 4.4  
130 \*/  
131 @Override  
132 public boolean removeIf(Predicate<? super E> filter) {  
133 throw new UnsupportedOperationException();  
134 }  
135  
136 @Override  
137 public boolean removeAll(final Collection<?> coll) {  
138 throw new UnsupportedOperationException();  
139 }  
140  
141 @Override  
142 public boolean retainAll(final Collection<?> coll) {  
143 throw new UnsupportedOperationException();  
144 }  
145  
146 //-----------------------------------------------------------------------  
147  
148 @Override  
149 public boolean offer(final E obj) {  
150 throw new UnsupportedOperationException();  
151 }  
152  
153 @Override  
154 public E poll() {  
155 throw new UnsupportedOperationException();  
156 }  
157  
158 @Override  
159 public E remove() {  
160 throw new UnsupportedOperationException();  
161 }  
162  
163}