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.util.Queue;  
020  
021import org.apache.commons.collections4.collection.SynchronizedCollection;  
022  
023/\*\*  
024 \* Decorates another {@link Queue} to synchronize its behaviour for a multi-threaded environment.  
025 \* <p>  
026 \* Methods are synchronized, then forwarded to the decorated queue. Iterators must be separately synchronized around the  
027 \* loop.  
028 \* </p>  
029 \*  
030 \* @param <E> the type of the elements in the collection  
031 \* @since 4.2  
032 \*/  
033public class SynchronizedQueue<E> extends SynchronizedCollection<E> implements Queue<E> {  
034  
035 /\*\* Serialization version \*/  
036 private static final long serialVersionUID = 1L;  
037  
038 /\*\*  
039 \* Factory method to create a synchronized queue.  
040 \*  
041 \* @param <E>  
042 \* the type of the elements in the queue  
043 \* @param queue  
044 \* the queue to decorate, must not be null  
045 \* @return a new synchronized Queue  
046 \* @throws NullPointerException  
047 \* if queue is null  
048 \*/  
049 public static <E> SynchronizedQueue<E> synchronizedQueue(final Queue<E> queue) {  
050 return new SynchronizedQueue<>(queue);  
051 }  
052  
053 // -----------------------------------------------------------------------  
054 /\*\*  
055 \* Constructor that wraps (not copies).  
056 \*  
057 \* @param queue  
058 \* the queue to decorate, must not be null  
059 \* @throws NullPointerException  
060 \* if queue is null  
061 \*/  
062 protected SynchronizedQueue(final Queue<E> queue) {  
063 super(queue);  
064 }  
065  
066 /\*\*  
067 \* Constructor that wraps (not copies).  
068 \*  
069 \* @param queue  
070 \* the queue to decorate, must not be null  
071 \* @param lock  
072 \* the lock to use, must not be null  
073 \* @throws NullPointerException  
074 \* if queue or lock is null  
075 \*/  
076 protected SynchronizedQueue(final Queue<E> queue, final Object lock) {  
077 super(queue, lock);  
078 }  
079  
080 /\*\*  
081 \* Gets the queue being decorated.  
082 \*  
083 \* @return the decorated queue  
084 \*/  
085 @Override  
086 protected Queue<E> decorated() {  
087 return (Queue<E>) super.decorated();  
088 }  
089  
090 @Override  
091 public E element() {  
092 synchronized (lock) {  
093 return decorated().element();  
094 }  
095 }  
096  
097 @Override  
098 public boolean equals(final Object object) {  
099 if (object == this) {  
100 return true;  
101 }  
102 synchronized (lock) {  
103 return decorated().equals(object);  
104 }  
105 }  
106  
107 // -----------------------------------------------------------------------  
108  
109 @Override  
110 public int hashCode() {  
111 synchronized (lock) {  
112 return decorated().hashCode();  
113 }  
114 }  
115  
116 @Override  
117 public boolean offer(final E e) {  
118 synchronized (lock) {  
119 return decorated().offer(e);  
120 }  
121 }  
122  
123 @Override  
124 public E peek() {  
125 synchronized (lock) {  
126 return decorated().peek();  
127 }  
128 }  
129  
130 @Override  
131 public E poll() {  
132 synchronized (lock) {  
133 return decorated().poll();  
134 }  
135 }  
136  
137 @Override  
138 public E remove() {  
139 synchronized (lock) {  
140 return decorated().remove();  
141 }  
142 }  
143  
144}