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;  
018  
019import java.util.LinkedList;  
020import java.util.Queue;  
021  
022import org.apache.commons.collections4.queue.PredicatedQueue;  
023import org.apache.commons.collections4.queue.SynchronizedQueue;  
024import org.apache.commons.collections4.queue.TransformedQueue;  
025import org.apache.commons.collections4.queue.UnmodifiableQueue;  
026  
027/\*\*  
028 \* Provides utility methods and decorators for {@link Queue} instances.  
029 \*  
030 \* @since 4.0  
031 \*/  
032public class QueueUtils {  
033  
034 /\*\*  
035 \* An empty unmodifiable queue.  
036 \*/  
037 @SuppressWarnings("rawtypes") // OK, empty queue is compatible with any type  
038 public static final Queue EMPTY\_QUEUE = UnmodifiableQueue.unmodifiableQueue(new LinkedList<>());  
039  
040 /\*\*  
041 \* <code>QueueUtils</code> should not normally be instantiated.  
042 \*/  
043 private QueueUtils() {}  
044  
045 //-----------------------------------------------------------------------  
046  
047 /\*\*  
048 \* Returns a synchronized (thread-safe) queue backed by the given queue.  
049 \* In order to guarantee serial access, it is critical that all access to the  
050 \* backing queue is accomplished through the returned queue.  
051 \* <p>  
052 \* It is imperative that the user manually synchronize on the returned queue  
053 \* when iterating over it:  
054 \*  
055 \* <pre>  
056 \* Queue queue = QueueUtils.synchronizedQueue(new CircularFifoQueue());  
057 \* ...  
058 \* synchronized(queue) {  
059 \* Iterator i = queue.iterator(); // Must be in synchronized block  
060 \* while (i.hasNext())  
061 \* foo(i.next());  
062 \* }  
063 \* }  
064 \* </pre>  
065 \*  
066 \* Failure to follow this advice may result in non-deterministic behavior.  
067 \*  
068 \* @param <E> the element type  
069 \* @param queue the queue to synchronize, must not be null  
070 \* @return a synchronized queue backed by that queue  
071 \* @throws NullPointerException if the queue is null  
072 \* @since 4.2  
073 \*/  
074 public static <E> Queue<E> synchronizedQueue(final Queue<E> queue) {  
075 return SynchronizedQueue.synchronizedQueue(queue);  
076 }  
077  
078 /\*\*  
079 \* Returns an unmodifiable queue backed by the given queue.  
080 \*  
081 \* @param <E> the type of the elements in the queue  
082 \* @param queue the queue to make unmodifiable, must not be null  
083 \* @return an unmodifiable queue backed by that queue  
084 \* @throws NullPointerException if the queue is null  
085 \*/  
086 public static <E> Queue<E> unmodifiableQueue(final Queue<? extends E> queue) {  
087 return UnmodifiableQueue.unmodifiableQueue(queue);  
088 }  
089  
090 /\*\*  
091 \* Returns a predicated (validating) queue backed by the given queue.  
092 \* <p>  
093 \* Only objects that pass the test in the given predicate can be added to the queue.  
094 \* Trying to add an invalid object results in an IllegalArgumentException.  
095 \* It is important not to use the original queue after invoking this method,  
096 \* as it is a backdoor for adding invalid objects.  
097 \*  
098 \* @param <E> the type of the elements in the queue  
099 \* @param queue the queue to predicate, must not be null  
100 \* @param predicate the predicate used to evaluate new elements, must not be null  
101 \* @return a predicated queue  
102 \* @throws NullPointerException if the queue or predicate is null  
103 \*/  
104 public static <E> Queue<E> predicatedQueue(final Queue<E> queue, final Predicate<? super E> predicate) {  
105 return PredicatedQueue.predicatedQueue(queue, predicate);  
106 }  
107  
108 /\*\*  
109 \* Returns a transformed queue backed by the given queue.  
110 \* <p>  
111 \* Each object is passed through the transformer as it is added to the  
112 \* Queue. It is important not to use the original queue after invoking this  
113 \* method, as it is a backdoor for adding untransformed objects.  
114 \* <p>  
115 \* Existing entries in the specified queue will not be transformed.  
116 \* If you want that behaviour, see {@link TransformedQueue#transformedQueue}.  
117 \*  
118 \* @param <E> the type of the elements in the queue  
119 \* @param queue the queue to predicate, must not be null  
120 \* @param transformer the transformer for the queue, must not be null  
121 \* @return a transformed queue backed by the given queue  
122 \* @throws NullPointerException if the queue or transformer is null  
123 \*/  
124 public static <E> Queue<E> transformingQueue(final Queue<E> queue,  
125 final Transformer<? super E, ? extends E> transformer) {  
126 return TransformedQueue.transformingQueue(queue, transformer);  
127 }  
128  
129 /\*\*  
130 \* Get an empty <code>Queue</code>.  
131 \*  
132 \* @param <E> the type of the elements in the queue  
133 \* @return an empty {@link Queue}  
134 \*/  
135 @SuppressWarnings("unchecked") // OK, empty queue is compatible with any type  
136 public static <E> Queue<E> emptyQueue() {  
137 return EMPTY\_QUEUE;  
138 }  
139}