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.Transformer;  
022import org.apache.commons.collections4.collection.TransformedCollection;  
023  
024/\*\*  
025 \* Decorates another {@link Queue} to transform objects that are added.  
026 \* <p>  
027 \* The add/offer methods are affected by this class.  
028 \* Thus objects must be removed or searched for using their transformed form.  
029 \* For example, if the transformation converts Strings to Integers, you must  
030 \* use the Integer form to remove objects.  
031 \* </p>  
032 \*  
033 \* @param <E> the type of elements held in this queue  
034 \* @since 4.0  
035 \*/  
036public class TransformedQueue<E> extends TransformedCollection<E> implements Queue<E> {  
037  
038 /\*\* Serialization version \*/  
039 private static final long serialVersionUID = -7901091318986132033L;  
040  
041 /\*\*  
042 \* Factory method to create a transforming queue.  
043 \* <p>  
044 \* If there are any elements already in the queue being decorated, they  
045 \* are NOT transformed.  
046 \* Contrast this with {@link #transformedQueue(Queue, Transformer)}.  
047 \*  
048 \* @param <E> the type of the elements in the queue  
049 \* @param queue the queue to decorate, must not be null  
050 \* @param transformer the transformer to use for conversion, must not be null  
051 \* @return a new transformed Queue  
052 \* @throws NullPointerException if queue or transformer is null  
053 \*/  
054 public static <E> TransformedQueue<E> transformingQueue(final Queue<E> queue,  
055 final Transformer<? super E, ? extends E> transformer) {  
056 return new TransformedQueue<>(queue, transformer);  
057 }  
058  
059 /\*\*  
060 \* Factory method to create a transforming queue that will transform  
061 \* existing contents of the specified queue.  
062 \* <p>  
063 \* If there are any elements already in the queue being decorated, they  
064 \* will be transformed by this method.  
065 \* Contrast this with {@link #transformingQueue(Queue, Transformer)}.  
066 \*  
067 \* @param <E> the type of the elements in the queue  
068 \* @param queue the queue to decorate, must not be null  
069 \* @param transformer the transformer to use for conversion, must not be null  
070 \* @return a new transformed Queue  
071 \* @throws NullPointerException if queue or transformer is null  
072 \* @since 4.0  
073 \*/  
074 public static <E> TransformedQueue<E> transformedQueue(final Queue<E> queue,  
075 final Transformer<? super E, ? extends E> transformer) {  
076 // throws IAE if queue or transformer is null  
077 final TransformedQueue<E> decorated = new TransformedQueue<>(queue, transformer);  
078 if (queue.size() > 0) {  
079 @SuppressWarnings("unchecked") // queue is type <E>  
080 final E[] values = (E[]) queue.toArray(); // NOPMD - false positive for generics  
081 queue.clear();  
082 for (final E value : values) {  
083 decorated.decorated().add(transformer.transform(value));  
084 }  
085 }  
086 return decorated;  
087 }  
088  
089 //-----------------------------------------------------------------------  
090 /\*\*  
091 \* Constructor that wraps (not copies).  
092 \* <p>  
093 \* If there are any elements already in the queue being decorated, they  
094 \* are NOT transformed.  
095 \*  
096 \* @param queue the queue to decorate, must not be null  
097 \* @param transformer the transformer to use for conversion, must not be null  
098 \* @throws NullPointerException if queue or transformer is null  
099 \*/  
100 protected TransformedQueue(final Queue<E> queue, final Transformer<? super E, ? extends E> transformer) {  
101 super(queue, transformer);  
102 }  
103  
104 /\*\*  
105 \* Gets the decorated queue.  
106 \*  
107 \* @return the decorated queue  
108 \*/  
109 protected Queue<E> getQueue() {  
110 return (Queue<E>) decorated();  
111 }  
112  
113 //-----------------------------------------------------------------------  
114  
115 @Override  
116 public boolean offer(final E obj) {  
117 return getQueue().offer(transform(obj));  
118 }  
119  
120 @Override  
121 public E poll() {  
122 return getQueue().poll();  
123 }  
124  
125 @Override  
126 public E peek() {  
127 return getQueue().peek();  
128 }  
129  
130 @Override  
131 public E element() {  
132 return getQueue().element();  
133 }  
134  
135 @Override  
136 public E remove() {  
137 return getQueue().remove();  
138 }  
139  
140}