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.Predicate;  
022import org.apache.commons.collections4.collection.PredicatedCollection;  
023  
024/\*\*  
025 \* Decorates another {@link Queue} to validate that additions  
026 \* match a specified predicate.  
027 \* <p>  
028 \* This queue exists to provide validation for the decorated queue.  
029 \* It is normally created to decorate an empty queue.  
030 \* If an object cannot be added to the queue, an IllegalArgumentException is thrown.  
031 \* </p>  
032 \* <p>  
033 \* One usage would be to ensure that no null entries are added to the queue.  
034 \* </p>  
035 \* <pre>Queue queue = PredicatedQueue.predicatedQueue(new UnboundedFifoQueue(), NotNullPredicate.INSTANCE);</pre>  
036 \*  
037 \* @param <E> the type of elements held in this queue  
038 \* @since 4.0  
039 \*/  
040public class PredicatedQueue<E> extends PredicatedCollection<E> implements Queue<E> {  
041  
042 /\*\* Serialization version \*/  
043 private static final long serialVersionUID = 2307609000539943581L;  
044  
045 /\*\*  
046 \* Factory method to create a predicated (validating) queue.  
047 \* <p>  
048 \* If there are any elements already in the queue being decorated, they  
049 \* are validated.  
050 \*  
051 \* @param <E> the type of the elements in the queue  
052 \* @param Queue the queue to decorate, must not be null  
053 \* @param predicate the predicate to use for validation, must not be null  
054 \* @return a new predicated queue  
055 \* @throws NullPointerException if queue or predicate is null  
056 \* @throws IllegalArgumentException if the queue contains invalid elements  
057 \*/  
058 public static <E> PredicatedQueue<E> predicatedQueue(final Queue<E> Queue,  
059 final Predicate<? super E> predicate) {  
060 return new PredicatedQueue<>(Queue, predicate);  
061 }  
062  
063 //-----------------------------------------------------------------------  
064 /\*\*  
065 \* Constructor that wraps (not copies).  
066 \* <p>  
067 \* If there are any elements already in the collection being decorated, they  
068 \* are validated.  
069 \*  
070 \* @param queue the queue to decorate, must not be null  
071 \* @param predicate the predicate to use for validation, must not be null  
072 \* @throws NullPointerException if queue or predicate is null  
073 \* @throws IllegalArgumentException if the Queue contains invalid elements  
074 \*/  
075 protected PredicatedQueue(final Queue<E> queue, final Predicate<? super E> predicate) {  
076 super(queue, predicate);  
077 }  
078  
079 /\*\*  
080 \* Gets the queue being decorated.  
081 \*  
082 \* @return the decorated queue  
083 \*/  
084 @Override  
085 protected Queue<E> decorated() {  
086 return (Queue<E>) super.decorated();  
087 }  
088  
089 //-----------------------------------------------------------------------  
090  
091 /\*\*  
092 \* Override to validate the object being added to ensure it matches  
093 \* the predicate.  
094 \*  
095 \* @param object the object being added  
096 \* @return the result of adding to the underlying queue  
097 \* @throws IllegalArgumentException if the add is invalid  
098 \*/  
099 @Override  
100 public boolean offer(final E object) {  
101 validate(object);  
102 return decorated().offer(object);  
103 }  
104  
105 @Override  
106 public E poll() {  
107 return decorated().poll();  
108 }  
109  
110 @Override  
111 public E peek() {  
112 return decorated().peek();  
113 }  
114  
115 @Override  
116 public E element() {  
117 return decorated().element();  
118 }  
119  
120 @Override  
121 public E remove() {  
122 return decorated().remove();  
123 }  
124  
125}