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.set;  
018  
019import java.util.Comparator;  
020import java.util.SortedSet;  
021  
022import org.apache.commons.collections4.Predicate;  
023  
024/\*\*  
025 \* Decorates another <code>SortedSet</code> to validate that all additions  
026 \* match a specified predicate.  
027 \* <p>  
028 \* This set exists to provide validation for the decorated set.  
029 \* It is normally created to decorate an empty set.  
030 \* If an object cannot be added to the set, an IllegalArgumentException is thrown.  
031 \* </p>  
032 \* <p>  
033 \* One usage would be to ensure that no null entries are added to the set.  
034 \* </p>  
035 \* <pre>  
036 \* SortedSet set =  
037 \* PredicatedSortedSet.predicatedSortedSet(new TreeSet(),  
038 \* NotNullPredicate.notNullPredicate());  
039 \* </pre>  
040 \* <p>  
041 \* This class is Serializable from Commons Collections 3.1.  
042 \* </p>  
043 \*  
044 \* @param <E> the type of the elements in this set  
045 \* @since 3.0  
046 \*/  
047public class PredicatedSortedSet<E> extends PredicatedSet<E> implements SortedSet<E> {  
048  
049 /\*\* Serialization version \*/  
050 private static final long serialVersionUID = -9110948148132275052L;  
051  
052 /\*\*  
053 \* Factory method to create a predicated (validating) sorted set.  
054 \* <p>  
055 \* If there are any elements already in the set being decorated, they  
056 \* are validated.  
057 \*  
058 \* @param <E> the element type  
059 \* @param set the set to decorate, must not be null  
060 \* @param predicate the predicate to use for validation, must not be null  
061 \* @return a new predicated sorted set.  
062 \* @throws NullPointerException if set or predicate is null  
063 \* @throws IllegalArgumentException if the set contains invalid elements  
064 \* @since 4.0  
065 \*/  
066 public static <E> PredicatedSortedSet<E> predicatedSortedSet(final SortedSet<E> set,  
067 final Predicate<? super E> predicate) {  
068 return new PredicatedSortedSet<>(set, predicate);  
069 }  
070  
071 //-----------------------------------------------------------------------  
072 /\*\*  
073 \* Constructor that wraps (not copies).  
074 \* <p>  
075 \* If there are any elements already in the set being decorated, they  
076 \* are validated.  
077 \*  
078 \* @param set the set to decorate, must not be null  
079 \* @param predicate the predicate to use for validation, must not be null  
080 \* @throws NullPointerException if set or predicate is null  
081 \* @throws IllegalArgumentException if the set contains invalid elements  
082 \*/  
083 protected PredicatedSortedSet(final SortedSet<E> set, final Predicate<? super E> predicate) {  
084 super(set, predicate);  
085 }  
086  
087 /\*\*  
088 \* Gets the sorted set being decorated.  
089 \*  
090 \* @return the decorated sorted set  
091 \*/  
092 @Override  
093 protected SortedSet<E> decorated() {  
094 return (SortedSet<E>) super.decorated();  
095 }  
096  
097 //-----------------------------------------------------------------------  
098 @Override  
099 public Comparator<? super E> comparator() {  
100 return decorated().comparator();  
101 }  
102  
103 @Override  
104 public E first() {  
105 return decorated().first();  
106 }  
107  
108 @Override  
109 public E last() {  
110 return decorated().last();  
111 }  
112  
113 @Override  
114 public SortedSet<E> subSet(final E fromElement, final E toElement) {  
115 final SortedSet<E> sub = decorated().subSet(fromElement, toElement);  
116 return new PredicatedSortedSet<>(sub, predicate);  
117 }  
118  
119 @Override  
120 public SortedSet<E> headSet(final E toElement) {  
121 final SortedSet<E> head = decorated().headSet(toElement);  
122 return new PredicatedSortedSet<>(head, predicate);  
123 }  
124  
125 @Override  
126 public SortedSet<E> tailSet(final E fromElement) {  
127 final SortedSet<E> tail = decorated().tailSet(fromElement);  
128 return new PredicatedSortedSet<>(tail, predicate);  
129 }  
130  
131}