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.map;  
018  
019import java.util.Comparator;  
020import java.util.SortedMap;  
021  
022import org.apache.commons.collections4.Predicate;  
023  
024/\*\*  
025 \* Decorates another <code>SortedMap</code> to validate that additions  
026 \* match a specified predicate.  
027 \* <p>  
028 \* This map exists to provide validation for the decorated map.  
029 \* It is normally created to decorate an empty map.  
030 \* If an object cannot be added to the map, an IllegalArgumentException is thrown.  
031 \* </p>  
032 \* <p>  
033 \* One usage would be to ensure that no null keys are added to the map.  
034 \* </p>  
035 \* <pre>  
036 \* SortedMap map =  
037 \* PredicatedSortedMap.predicatedSortedMap(new TreeMap(),  
038 \* NotNullPredicate.notNullPredicate(),  
039 \* null);  
040 \* </pre>  
041 \* <p>  
042 \* <strong>Note that PredicatedSortedMap is not synchronized and is not thread-safe.</strong>  
043 \* If you wish to use this map from multiple threads concurrently, you must use  
044 \* appropriate synchronization. The simplest approach is to wrap this map  
045 \* using {@link java.util.Collections#synchronizedSortedMap}. This class may throw  
046 \* exceptions when accessed by concurrent threads without synchronization.  
047 \* </p>  
048 \* <p>  
049 \* This class is Serializable from Commons Collections 3.1.  
050 \* </p>  
051 \*  
052 \* @param <K> the type of the keys in this map  
053 \* @param <V> the type of the values in this map  
054 \* @since 3.0  
055 \*/  
056public class PredicatedSortedMap<K, V> extends PredicatedMap<K, V> implements SortedMap<K, V> {  
057  
058 /\*\* Serialization version \*/  
059 private static final long serialVersionUID = 3359846175935304332L;  
060  
061 /\*\*  
062 \* Factory method to create a predicated (validating) sorted map.  
063 \* <p>  
064 \* If there are any elements already in the list being decorated, they  
065 \* are validated.  
066 \*  
067 \* @param <K> the key type  
068 \* @param <V> the value type  
069 \* @param map the map to decorate, must not be null  
070 \* @param keyPredicate the predicate to validate the keys, null means no check  
071 \* @param valuePredicate the predicate to validate to values, null means no check  
072 \* @return a new predicated sorted map  
073 \* @throws NullPointerException if the map is null  
074 \* @since 4.0  
075 \*/  
076 public static <K, V> PredicatedSortedMap<K, V> predicatedSortedMap(final SortedMap<K, V> map,  
077 final Predicate<? super K> keyPredicate, final Predicate<? super V> valuePredicate) {  
078 return new PredicatedSortedMap<>(map, keyPredicate, valuePredicate);  
079 }  
080  
081 //-----------------------------------------------------------------------  
082 /\*\*  
083 \* Constructor that wraps (not copies).  
084 \*  
085 \* @param map the map to decorate, must not be null  
086 \* @param keyPredicate the predicate to validate the keys, null means no check  
087 \* @param valuePredicate the predicate to validate to values, null means no check  
088 \* @throws NullPointerException if the map is null  
089 \*/  
090 protected PredicatedSortedMap(final SortedMap<K, V> map, final Predicate<? super K> keyPredicate,  
091 final Predicate<? super V> valuePredicate) {  
092 super(map, keyPredicate, valuePredicate);  
093 }  
094  
095 //-----------------------------------------------------------------------  
096 /\*\*  
097 \* Gets the map being decorated.  
098 \*  
099 \* @return the decorated map  
100 \*/  
101 protected SortedMap<K, V> getSortedMap() {  
102 return (SortedMap<K, V>) map;  
103 }  
104  
105 //-----------------------------------------------------------------------  
106 @Override  
107 public K firstKey() {  
108 return getSortedMap().firstKey();  
109 }  
110  
111 @Override  
112 public K lastKey() {  
113 return getSortedMap().lastKey();  
114 }  
115  
116 @Override  
117 public Comparator<? super K> comparator() {  
118 return getSortedMap().comparator();  
119 }  
120  
121 @Override  
122 public SortedMap<K, V> subMap(final K fromKey, final K toKey) {  
123 final SortedMap<K, V> map = getSortedMap().subMap(fromKey, toKey);  
124 return new PredicatedSortedMap<>(map, keyPredicate, valuePredicate);  
125 }  
126  
127 @Override  
128 public SortedMap<K, V> headMap(final K toKey) {  
129 final SortedMap<K, V> map = getSortedMap().headMap(toKey);  
130 return new PredicatedSortedMap<>(map, keyPredicate, valuePredicate);  
131 }  
132  
133 @Override  
134 public SortedMap<K, V> tailMap(final K fromKey) {  
135 final SortedMap<K, V> map = getSortedMap().tailMap(fromKey);  
136 return new PredicatedSortedMap<>(map, keyPredicate, valuePredicate);  
137 }  
138  
139}