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.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.io.Serializable;  
023import java.util.Collection;  
024import java.util.Map;  
025import java.util.Set;  
026import java.util.SortedMap;  
027  
028import org.apache.commons.collections4.BoundedMap;  
029import org.apache.commons.collections4.CollectionUtils;  
030import org.apache.commons.collections4.collection.UnmodifiableCollection;  
031import org.apache.commons.collections4.set.UnmodifiableSet;  
032  
033/\*\*  
034 \* Decorates another <code>SortedMap</code> to fix the size blocking add/remove.  
035 \* <p>  
036 \* Any action that would change the size of the map is disallowed.  
037 \* The put method is allowed to change the value associated with an existing  
038 \* key however.  
039 \* </p>  
040 \* <p>  
041 \* If trying to remove or clear the map, an UnsupportedOperationException is  
042 \* thrown. If trying to put a new mapping into the map, an  
043 \* IllegalArgumentException is thrown. This is because the put method can  
044 \* succeed if the mapping's key already exists in the map, so the put method  
045 \* is not always unsupported.  
046 \* </p>  
047 \* <p>  
048 \* <strong>Note that FixedSizeSortedMap is not synchronized and is not thread-safe.</strong>  
049 \* If you wish to use this map from multiple threads concurrently, you must use  
050 \* appropriate synchronization. The simplest approach is to wrap this map  
051 \* using {@link java.util.Collections#synchronizedSortedMap}. This class may throw  
052 \* exceptions when accessed by concurrent threads without synchronization.  
053 \* </p>  
054 \* <p>  
055 \* This class is Serializable from Commons Collections 3.1.  
056 \* </p>  
057 \*  
058 \* @param <K> the type of the keys in this map  
059 \* @param <V> the type of the values in this map  
060 \* @since 3.0  
061 \*/  
062public class FixedSizeSortedMap<K, V>  
063 extends AbstractSortedMapDecorator<K, V>  
064 implements BoundedMap<K, V>, Serializable {  
065  
066 /\*\* Serialization version \*/  
067 private static final long serialVersionUID = 3126019624511683653L;  
068  
069 /\*\*  
070 \* Factory method to create a fixed size sorted map.  
071 \*  
072 \* @param <K> the key type  
073 \* @param <V> the value type  
074 \* @param map the map to decorate, must not be null  
075 \* @return a new fixed size sorted map  
076 \* @throws NullPointerException if map is null  
077 \* @since 4.0  
078 \*/  
079 public static <K, V> FixedSizeSortedMap<K, V> fixedSizeSortedMap(final SortedMap<K, V> map) {  
080 return new FixedSizeSortedMap<>(map);  
081 }  
082  
083 //-----------------------------------------------------------------------  
084 /\*\*  
085 \* Constructor that wraps (not copies).  
086 \*  
087 \* @param map the map to decorate, must not be null  
088 \* @throws NullPointerException if map is null  
089 \*/  
090 protected FixedSizeSortedMap(final SortedMap<K, V> map) {  
091 super(map);  
092 }  
093  
094 /\*\*  
095 \* Gets the map being decorated.  
096 \*  
097 \* @return the decorated map  
098 \*/  
099 protected SortedMap<K, V> getSortedMap() {  
100 return (SortedMap<K, V>) map;  
101 }  
102  
103 //-----------------------------------------------------------------------  
104 /\*\*  
105 \* Write the map out using a custom routine.  
106 \*  
107 \* @param out the output stream  
108 \* @throws IOException if an error occurs while writing to the stream  
109 \*/  
110 private void writeObject(final ObjectOutputStream out) throws IOException {  
111 out.defaultWriteObject();  
112 out.writeObject(map);  
113 }  
114  
115 /\*\*  
116 \* Read the map in using a custom routine.  
117 \*  
118 \* @param in the input stream  
119 \* @throws IOException if an error occurs while reading from the stream  
120 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
121 \*/  
122 @SuppressWarnings("unchecked") // (1) should only fail if input stream is incorrect  
123 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
124 in.defaultReadObject();  
125 map = (Map<K, V>) in.readObject(); // (1)  
126 }  
127  
128 //-----------------------------------------------------------------------  
129 @Override  
130 public V put(final K key, final V value) {  
131 if (map.containsKey(key) == false) {  
132 throw new IllegalArgumentException("Cannot put new key/value pair - Map is fixed size");  
133 }  
134 return map.put(key, value);  
135 }  
136  
137 @Override  
138 public void putAll(final Map<? extends K, ? extends V> mapToCopy) {  
139 if (CollectionUtils.isSubCollection(mapToCopy.keySet(), keySet())) {  
140 throw new IllegalArgumentException("Cannot put new key/value pair - Map is fixed size");  
141 }  
142 map.putAll(mapToCopy);  
143 }  
144  
145 @Override  
146 public void clear() {  
147 throw new UnsupportedOperationException("Map is fixed size");  
148 }  
149  
150 @Override  
151 public V remove(final Object key) {  
152 throw new UnsupportedOperationException("Map is fixed size");  
153 }  
154  
155 @Override  
156 public Set<Map.Entry<K, V>> entrySet() {  
157 return UnmodifiableSet.unmodifiableSet(map.entrySet());  
158 }  
159  
160 @Override  
161 public Set<K> keySet() {  
162 return UnmodifiableSet.unmodifiableSet(map.keySet());  
163 }  
164  
165 @Override  
166 public Collection<V> values() {  
167 return UnmodifiableCollection.unmodifiableCollection(map.values());  
168 }  
169  
170 //-----------------------------------------------------------------------  
171 @Override  
172 public SortedMap<K, V> subMap(final K fromKey, final K toKey) {  
173 return new FixedSizeSortedMap<>(getSortedMap().subMap(fromKey, toKey));  
174 }  
175  
176 @Override  
177 public SortedMap<K, V> headMap(final K toKey) {  
178 return new FixedSizeSortedMap<>(getSortedMap().headMap(toKey));  
179 }  
180  
181 @Override  
182 public SortedMap<K, V> tailMap(final K fromKey) {  
183 return new FixedSizeSortedMap<>(getSortedMap().tailMap(fromKey));  
184 }  
185  
186 @Override  
187 public boolean isFull() {  
188 return true;  
189 }  
190  
191 @Override  
192 public int maxSize() {  
193 return size();  
194 }  
195  
196}