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.trie;  
018  
019import java.io.Serializable;  
020import java.util.Collection;  
021import java.util.Collections;  
022import java.util.Comparator;  
023import java.util.Map;  
024import java.util.Set;  
025import java.util.SortedMap;  
026  
027import org.apache.commons.collections4.OrderedMapIterator;  
028import org.apache.commons.collections4.Trie;  
029import org.apache.commons.collections4.Unmodifiable;  
030import org.apache.commons.collections4.iterators.UnmodifiableOrderedMapIterator;  
031  
032/\*\*  
033 \* An unmodifiable {@link Trie}.  
034 \*  
035 \* @param <K> the type of the keys in this map  
036 \* @param <V> the type of the values in this map  
037 \* @since 4.0  
038 \*/  
039public class UnmodifiableTrie<K, V> implements Trie<K, V>, Serializable, Unmodifiable {  
040  
041 /\*\* Serialization version \*/  
042 private static final long serialVersionUID = -7156426030315945159L;  
043  
044 private final Trie<K, V> delegate;  
045  
046 /\*\*  
047 \* Factory method to create a unmodifiable trie.  
048 \*  
049 \* @param <K> the key type  
050 \* @param <V> the value type  
051 \* @param trie the trie to decorate, must not be null  
052 \* @return a new unmodifiable trie  
053 \* @throws NullPointerException if trie is null  
054 \*/  
055 public static <K, V> Trie<K, V> unmodifiableTrie(final Trie<K, ? extends V> trie) {  
056 if (trie instanceof Unmodifiable) {  
057 @SuppressWarnings("unchecked") // safe to upcast  
058 final Trie<K, V> tmpTrie = (Trie<K, V>) trie;  
059 return tmpTrie;  
060 }  
061 return new UnmodifiableTrie<>(trie);  
062 }  
063  
064 //-----------------------------------------------------------------------  
065 /\*\*  
066 \* Constructor that wraps (not copies).  
067 \*  
068 \* @param trie the trie to decorate, must not be null  
069 \* @throws NullPointerException if trie is null  
070 \*/  
071 public UnmodifiableTrie(final Trie<K, ? extends V> trie) {  
072 if (trie == null) {  
073 throw new NullPointerException("Trie must not be null");  
074 }  
075 @SuppressWarnings("unchecked") // safe to upcast  
076 final Trie<K, V> tmpTrie = (Trie<K, V>) trie;  
077 this.delegate = tmpTrie;  
078 }  
079  
080 //-----------------------------------------------------------------------  
081  
082 @Override  
083 public Set<Entry<K, V>> entrySet() {  
084 return Collections.unmodifiableSet(delegate.entrySet());  
085 }  
086  
087 @Override  
088 public Set<K> keySet() {  
089 return Collections.unmodifiableSet(delegate.keySet());  
090 }  
091  
092 @Override  
093 public Collection<V> values() {  
094 return Collections.unmodifiableCollection(delegate.values());  
095 }  
096  
097 @Override  
098 public void clear() {  
099 throw new UnsupportedOperationException();  
100 }  
101  
102 @Override  
103 public boolean containsKey(final Object key) {  
104 return delegate.containsKey(key);  
105 }  
106  
107 @Override  
108 public boolean containsValue(final Object value) {  
109 return delegate.containsValue(value);  
110 }  
111  
112 @Override  
113 public V get(final Object key) {  
114 return delegate.get(key);  
115 }  
116  
117 @Override  
118 public boolean isEmpty() {  
119 return delegate.isEmpty();  
120 }  
121  
122 @Override  
123 public V put(final K key, final V value) {  
124 throw new UnsupportedOperationException();  
125 }  
126  
127 @Override  
128 public void putAll(final Map<? extends K, ? extends V> m) {  
129 throw new UnsupportedOperationException();  
130 }  
131  
132 @Override  
133 public V remove(final Object key) {  
134 throw new UnsupportedOperationException();  
135 }  
136  
137 @Override  
138 public int size() {  
139 return delegate.size();  
140 }  
141  
142 @Override  
143 public K firstKey() {  
144 return delegate.firstKey();  
145 }  
146  
147 @Override  
148 public SortedMap<K, V> headMap(final K toKey) {  
149 return Collections.unmodifiableSortedMap(delegate.headMap(toKey));  
150 }  
151  
152 @Override  
153 public K lastKey() {  
154 return delegate.lastKey();  
155 }  
156  
157 @Override  
158 public SortedMap<K, V> subMap(final K fromKey, final K toKey) {  
159 return Collections.unmodifiableSortedMap(delegate.subMap(fromKey, toKey));  
160 }  
161  
162 @Override  
163 public SortedMap<K, V> tailMap(final K fromKey) {  
164 return Collections.unmodifiableSortedMap(delegate.tailMap(fromKey));  
165 }  
166  
167 @Override  
168 public SortedMap<K, V> prefixMap(final K key) {  
169 return Collections.unmodifiableSortedMap(delegate.prefixMap(key));  
170 }  
171  
172 @Override  
173 public Comparator<? super K> comparator() {  
174 return delegate.comparator();  
175 }  
176  
177 //-----------------------------------------------------------------------  
178 @Override  
179 public OrderedMapIterator<K, V> mapIterator() {  
180 final OrderedMapIterator<K, V> it = delegate.mapIterator();  
181 return UnmodifiableOrderedMapIterator.unmodifiableOrderedMapIterator(it);  
182 }  
183  
184 @Override  
185 public K nextKey(final K key) {  
186 return delegate.nextKey(key);  
187 }  
188  
189 @Override  
190 public K previousKey(final K key) {  
191 return delegate.previousKey(key);  
192 }  
193  
194 //-----------------------------------------------------------------------  
195 @Override  
196 public int hashCode() {  
197 return delegate.hashCode();  
198 }  
199  
200 @Override  
201 public boolean equals(final Object obj) {  
202 return delegate.equals(obj);  
203 }  
204  
205 @Override  
206 public String toString() {  
207 return delegate.toString();  
208 }  
209  
210}