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;  
018  
019import java.util.Collection;  
020import java.util.Map;  
021import java.util.Set;  
022  
023import org.apache.commons.collections4.collection.UnmodifiableCollection;  
024import org.apache.commons.collections4.iterators.UnmodifiableMapIterator;  
025import org.apache.commons.collections4.map.EntrySetToMapIteratorAdapter;  
026import org.apache.commons.collections4.map.UnmodifiableEntrySet;  
027import org.apache.commons.collections4.set.UnmodifiableSet;  
028  
029/\*\*  
030 \* Utilities for working with "split maps:" objects that implement {@link Put}  
031 \* and/or {@link Get} but not {@link Map}.  
032 \*  
033 \* @since 4.0  
034 \*  
035 \* @see Get  
036 \* @see Put  
037 \*/  
038public class SplitMapUtils {  
039  
040 /\*\*  
041 \* <code>SplitMapUtils</code> should not normally be instantiated.  
042 \*/  
043 private SplitMapUtils() {}  
044  
045 //-----------------------------------------------------------------------  
046  
047 private static class WrappedGet<K, V> implements IterableMap<K, V>, Unmodifiable {  
048 private final Get<K, V> get;  
049  
050 private WrappedGet(final Get<K, V> get) {  
051 this.get = get;  
052 }  
053  
054 @Override  
055 public void clear() {  
056 throw new UnsupportedOperationException();  
057 }  
058  
059 @Override  
060 public boolean containsKey(final Object key) {  
061 return get.containsKey(key);  
062 }  
063  
064 @Override  
065 public boolean containsValue(final Object value) {  
066 return get.containsValue(value);  
067 }  
068  
069 @Override  
070 public Set<Map.Entry<K, V>> entrySet() {  
071 return UnmodifiableEntrySet.unmodifiableEntrySet(get.entrySet());  
072 }  
073  
074 @Override  
075 public boolean equals(final Object arg0) {  
076 if (arg0 == this) {  
077 return true;  
078 }  
079 return arg0 instanceof WrappedGet && ((WrappedGet<?, ?>) arg0).get.equals(this.get);  
080 }  
081  
082 @Override  
083 public V get(final Object key) {  
084 return get.get(key);  
085 }  
086  
087 @Override  
088 public int hashCode() {  
089 return ("WrappedGet".hashCode() << 4) | get.hashCode();  
090 }  
091  
092 @Override  
093 public boolean isEmpty() {  
094 return get.isEmpty();  
095 }  
096  
097 @Override  
098 public Set<K> keySet() {  
099 return UnmodifiableSet.unmodifiableSet(get.keySet());  
100 }  
101  
102 @Override  
103 public V put(final K key, final V value) {  
104 throw new UnsupportedOperationException();  
105 }  
106  
107 @Override  
108 public void putAll(final Map<? extends K, ? extends V> t) {  
109 throw new UnsupportedOperationException();  
110 }  
111  
112 @Override  
113 public V remove(final Object key) {  
114 return get.remove(key);  
115 }  
116  
117 @Override  
118 public int size() {  
119 return get.size();  
120 }  
121  
122 @Override  
123 public Collection<V> values() {  
124 return UnmodifiableCollection.unmodifiableCollection(get.values());  
125 }  
126  
127 @Override  
128 public MapIterator<K, V> mapIterator() {  
129 MapIterator<K, V> it;  
130 if (get instanceof IterableGet) {  
131 it = ((IterableGet<K, V>) get).mapIterator();  
132 } else {  
133 it = new EntrySetToMapIteratorAdapter<>(get.entrySet());  
134 }  
135 return UnmodifiableMapIterator.unmodifiableMapIterator(it);  
136 }  
137 }  
138  
139 private static class WrappedPut<K, V> implements Map<K, V>, Put<K, V> {  
140 private final Put<K, V> put;  
141  
142 private WrappedPut(final Put<K, V> put) {  
143 this.put = put;  
144 }  
145  
146 @Override  
147 public void clear() {  
148 put.clear();  
149 }  
150  
151 @Override  
152 public boolean containsKey(final Object key) {  
153 throw new UnsupportedOperationException();  
154 }  
155  
156 @Override  
157 public boolean containsValue(final Object value) {  
158 throw new UnsupportedOperationException();  
159 }  
160  
161 @Override  
162 public Set<Map.Entry<K, V>> entrySet() {  
163 throw new UnsupportedOperationException();  
164 }  
165  
166 @Override  
167 public boolean equals(final Object obj) {  
168 if (obj == this) {  
169 return true;  
170 }  
171 return obj instanceof WrappedPut && ((WrappedPut<?, ?>) obj).put.equals(this.put);  
172 }  
173  
174 @Override  
175 public V get(final Object key) {  
176 throw new UnsupportedOperationException();  
177 }  
178  
179 @Override  
180 public int hashCode() {  
181 return ("WrappedPut".hashCode() << 4) | put.hashCode();  
182 }  
183  
184 @Override  
185 public boolean isEmpty() {  
186 throw new UnsupportedOperationException();  
187 }  
188  
189 @Override  
190 public Set<K> keySet() {  
191 throw new UnsupportedOperationException();  
192 }  
193  
194 @Override  
195 @SuppressWarnings("unchecked")  
196 public V put(final K key, final V value) {  
197 return (V) put.put(key, value);  
198 }  
199  
200 @Override  
201 public void putAll(final Map<? extends K, ? extends V> t) {  
202 put.putAll(t);  
203 }  
204  
205 @Override  
206 public V remove(final Object key) {  
207 throw new UnsupportedOperationException();  
208 }  
209  
210 @Override  
211 public int size() {  
212 throw new UnsupportedOperationException();  
213 }  
214  
215 @Override  
216 public Collection<V> values() {  
217 throw new UnsupportedOperationException();  
218 }  
219 }  
220  
221 //-----------------------------------------------------------------------  
222  
223 /\*\*  
224 \* Get the specified {@link Get} as an instance of {@link IterableMap}.  
225 \* If <code>get</code> implements {@link IterableMap} directly, no conversion will take place.  
226 \* If <code>get</code> implements {@link Map} but not {@link IterableMap} it will be decorated.  
227 \* Otherwise an {@link Unmodifiable} {@link IterableMap} will be returned.  
228 \* @param <K> the key type  
229 \* @param <V> the value type  
230 \* @param get to wrap, must not be null  
231 \* @return {@link IterableMap}  
232 \* @throws NullPointerException if the argument is null  
233 \*/  
234 @SuppressWarnings("unchecked")  
235 public static <K, V> IterableMap<K, V> readableMap(final Get<K, V> get) {  
236 if (get == null) {  
237 throw new NullPointerException("Get must not be null");  
238 }  
239 if (get instanceof Map) {  
240 return get instanceof IterableMap ?  
241 ((IterableMap<K, V>) get) :  
242 MapUtils.iterableMap((Map<K, V>) get);  
243 }  
244 return new WrappedGet<>(get);  
245 }  
246  
247 /\*\*  
248 \* Get the specified {@link Put} as an instanceof {@link Map}.  
249 \* If <code>put</code> implements {@link Map} directly, no conversion will take place.  
250 \* Otherwise a <em>write-only</em> {@link Map} will be returned. On such a {@link Map}  
251 \* it is recommended that the result of #put(K, V) be discarded as it likely will not  
252 \* match <code>V</code> at runtime.  
253 \*  
254 \* @param <K> the key type  
255 \* @param <V> the element type  
256 \* @param put to wrap, must not be null  
257 \* @return {@link Map}  
258 \* @throws NullPointerException if the argument is null  
259 \*/  
260 @SuppressWarnings("unchecked")  
261 public static <K, V> Map<K, V> writableMap(final Put<K, V> put) {  
262 if (put == null) {  
263 throw new NullPointerException("Put must not be null");  
264 }  
265 if (put instanceof Map) {  
266 return (Map<K, V>) put;  
267 }  
268 return new WrappedPut<>(put);  
269 }  
270  
271}