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.ArrayList;  
020import java.util.Collection;  
021import java.util.HashSet;  
022import java.util.List;  
023import java.util.Set;  
024  
025import org.apache.commons.collections4.bag.HashBag;  
026import org.apache.commons.collections4.multimap.ArrayListValuedHashMap;  
027import org.apache.commons.collections4.multimap.HashSetValuedHashMap;  
028import org.apache.commons.collections4.multimap.TransformedMultiValuedMap;  
029import org.apache.commons.collections4.multimap.UnmodifiableMultiValuedMap;  
030  
031/\*\*  
032 \* Provides utility methods and decorators for {@link MultiValuedMap} instances.  
033 \* <p>  
034 \* It contains various type safe and null safe methods. Additionally, it provides  
035 \* the following decorators:  
036 \* </p>  
037 \* <ul>  
038 \* <li>{@link #unmodifiableMultiValuedMap(MultiValuedMap)}</li>  
039 \* <li>{@link #transformedMultiValuedMap(MultiValuedMap, Transformer, Transformer)}</li>  
040 \* </ul>  
041 \*  
042 \* @since 4.1  
043 \*/  
044public class MultiMapUtils {  
045  
046 /\*\*  
047 \* <code>MultiMapUtils</code> should not normally be instantiated.  
048 \*/  
049 private MultiMapUtils() {}  
050  
051 /\*\*  
052 \* An empty {@link UnmodifiableMultiValuedMap}.  
053 \*/  
054 @SuppressWarnings({ "rawtypes", "unchecked" })  
055 public static final MultiValuedMap EMPTY\_MULTI\_VALUED\_MAP =  
056 UnmodifiableMultiValuedMap.unmodifiableMultiValuedMap(new ArrayListValuedHashMap(0, 0));  
057  
058 /\*\*  
059 \* Returns immutable EMPTY\_MULTI\_VALUED\_MAP with generic type safety.  
060 \*  
061 \* @param <K> the type of key in the map  
062 \* @param <V> the type of value in the map  
063 \* @return immutable and empty <code>MultiValuedMap</code>  
064 \*/  
065 @SuppressWarnings("unchecked")  
066 public static <K, V> MultiValuedMap<K, V> emptyMultiValuedMap() {  
067 return EMPTY\_MULTI\_VALUED\_MAP;  
068 }  
069  
070 // Null safe methods  
071  
072 /\*\*  
073 \* Returns an immutable empty <code>MultiValuedMap</code> if the argument is  
074 \* <code>null</code>, or the argument itself otherwise.  
075 \*  
076 \* @param <K> the type of key in the map  
077 \* @param <V> the type of value in the map  
078 \* @param map the map, may be null  
079 \* @return an empty {@link MultiValuedMap} if the argument is null  
080 \*/  
081 @SuppressWarnings("unchecked")  
082 public static <K, V> MultiValuedMap<K, V> emptyIfNull(final MultiValuedMap<K, V> map) {  
083 return map == null ? EMPTY\_MULTI\_VALUED\_MAP : map;  
084 }  
085  
086 /\*\*  
087 \* Null-safe check if the specified <code>MultiValuedMap</code> is empty.  
088 \* <p>  
089 \* If the provided map is null, returns true.  
090 \*  
091 \* @param map the map to check, may be null  
092 \* @return true if the map is empty or null  
093 \*/  
094 public static boolean isEmpty(final MultiValuedMap<?, ?> map) {  
095 return map == null || map.isEmpty();  
096 }  
097  
098 // Null safe getters  
099 // -------------------------------------------------------------------------  
100  
101 /\*\*  
102 \* Gets a Collection from <code>MultiValuedMap</code> in a null-safe manner.  
103 \*  
104 \* @param <K> the key type  
105 \* @param <V> the value type  
106 \* @param map the {@link MultiValuedMap} to use  
107 \* @param key the key to look up  
108 \* @return the Collection in the {@link MultiValuedMap}, or null if input map is null  
109 \*/  
110 public static <K, V> Collection<V> getCollection(final MultiValuedMap<K, V> map, final K key) {  
111 if (map != null) {  
112 return map.get(key);  
113 }  
114 return null;  
115 }  
116  
117 // TODO: review the getValuesAsXXX methods - depending on the actual MultiValuedMap type, changes  
118 // to the returned collection might update the backing map. This should be clarified and/or prevented.  
119  
120 /\*\*  
121 \* Gets a List from <code>MultiValuedMap</code> in a null-safe manner.  
122 \*  
123 \* @param <K> the key type  
124 \* @param <V> the value type  
125 \* @param map the {@link MultiValuedMap} to use  
126 \* @param key the key to look up  
127 \* @return the Collection in the {@link MultiValuedMap} as List, or null if input map is null  
128 \*/  
129 public static <K, V> List<V> getValuesAsList(final MultiValuedMap<K, V> map, final K key) {  
130 if (map != null) {  
131 final Collection<V> col = map.get(key);  
132 if (col instanceof List) {  
133 return (List<V>) col;  
134 }  
135 return new ArrayList<>(col);  
136 }  
137 return null;  
138 }  
139  
140 /\*\*  
141 \* Gets a Set from <code>MultiValuedMap</code> in a null-safe manner.  
142 \*  
143 \* @param <K> the key type  
144 \* @param <V> the value type  
145 \* @param map the {@link MultiValuedMap} to use  
146 \* @param key the key to look up  
147 \* @return the Collection in the {@link MultiValuedMap} as Set, or null if input map is null  
148 \*/  
149 public static <K, V> Set<V> getValuesAsSet(final MultiValuedMap<K, V> map, final K key) {  
150 if (map != null) {  
151 final Collection<V> col = map.get(key);  
152 if (col instanceof Set) {  
153 return (Set<V>) col;  
154 }  
155 return new HashSet<>(col);  
156 }  
157 return null;  
158 }  
159  
160 /\*\*  
161 \* Gets a Bag from <code>MultiValuedMap</code> in a null-safe manner.  
162 \*  
163 \* @param <K> the key type  
164 \* @param <V> the value type  
165 \* @param map the {@link MultiValuedMap} to use  
166 \* @param key the key to look up  
167 \* @return the Collection in the {@link MultiValuedMap} as Bag, or null if input map is null  
168 \*/  
169 public static <K, V> Bag<V> getValuesAsBag(final MultiValuedMap<K, V> map, final K key) {  
170 if (map != null) {  
171 final Collection<V> col = map.get(key);  
172 if (col instanceof Bag) {  
173 return (Bag<V>) col;  
174 }  
175 return new HashBag<>(col);  
176 }  
177 return null;  
178 }  
179  
180 // Factory Methods  
181 // -----------------------------------------------------------------------  
182  
183 /\*\*  
184 \* Creates a {@link ListValuedMap} with an {@link java.util.ArrayList ArrayList} as  
185 \* collection class to store the values mapped to a key.  
186 \*  
187 \* @param <K> the key type  
188 \* @param <V> the value type  
189 \* @return a new <code>ListValuedMap</code>  
190 \*/  
191 public static <K, V> ListValuedMap<K, V> newListValuedHashMap() {  
192 return new ArrayListValuedHashMap<>();  
193 }  
194  
195 /\*\*  
196 \* Creates a {@link SetValuedMap} with an {@link java.util.HashSet HashSet} as  
197 \* collection class to store the values mapped to a key.  
198 \*  
199 \* @param <K> the key type  
200 \* @param <V> the value type  
201 \* @return a new {@link SetValuedMap}  
202 \*/  
203 public static <K, V> SetValuedMap<K, V> newSetValuedHashMap() {  
204 return new HashSetValuedHashMap<>();  
205 }  
206  
207 // MultiValuedMap Decorators  
208 // -----------------------------------------------------------------------  
209  
210 /\*\*  
211 \* Returns an <code>UnmodifiableMultiValuedMap</code> backed by the given  
212 \* map.  
213 \*  
214 \* @param <K> the key type  
215 \* @param <V> the value type  
216 \* @param map the {@link MultiValuedMap} to decorate, must not be null  
217 \* @return an unmodifiable {@link MultiValuedMap} backed by the provided map  
218 \* @throws NullPointerException if map is null  
219 \*/  
220 public static <K, V> MultiValuedMap<K, V> unmodifiableMultiValuedMap(  
221 final MultiValuedMap<? extends K, ? extends V> map) {  
222 return UnmodifiableMultiValuedMap.<K, V>unmodifiableMultiValuedMap(map);  
223 }  
224  
225 /\*\*  
226 \* Returns a <code>TransformedMultiValuedMap</code> backed by the given map.  
227 \* <p>  
228 \* This method returns a new <code>MultiValuedMap</code> (decorating the  
229 \* specified map) that will transform any new entries added to it. Existing  
230 \* entries in the specified map will not be transformed. If you want that  
231 \* behaviour, see {@link TransformedMultiValuedMap#transformedMap}.  
232 \* <p>  
233 \* Each object is passed through the transformers as it is added to the Map.  
234 \* It is important not to use the original map after invoking this method,  
235 \* as it is a back door for adding untransformed objects.  
236 \* <p>  
237 \* If there are any elements already in the map being decorated, they are  
238 \* NOT transformed.  
239 \*  
240 \* @param <K> the key type  
241 \* @param <V> the value type  
242 \* @param map the {@link MultiValuedMap} to transform, must not be null, typically empty  
243 \* @param keyTransformer the transformer for the map keys, null means no transformation  
244 \* @param valueTransformer the transformer for the map values, null means no transformation  
245 \* @return a transformed <code>MultiValuedMap</code> backed by the given map  
246 \* @throws NullPointerException if map is null  
247 \*/  
248 public static <K, V> MultiValuedMap<K, V> transformedMultiValuedMap(final MultiValuedMap<K, V> map,  
249 final Transformer<? super K, ? extends K> keyTransformer,  
250 final Transformer<? super V, ? extends V> valueTransformer) {  
251 return TransformedMultiValuedMap.transformingMap(map, keyTransformer, valueTransformer);  
252 }  
253  
254}