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.Map.Entry;  
022import java.util.Set;  
023  
024/\*\*  
025 \* Defines a map that holds a collection of values against each key.  
026 \* <p>  
027 \* A {@code MultiValuedMap} is a Map with slightly different semantics:  
028 \* </p>  
029 \* <ul>  
030 \* <li>Putting a value into the map will add the value to a {@link Collection} at that key.</li>  
031 \* <li>Getting a value will return a {@link Collection}, holding all the values put to that key.</li>  
032 \* </ul>  
033 \* <p>  
034 \* For example:  
035 \* </p>  
036 \* <pre>  
037 \* MultiValuedMap<K, String> map = new MultiValuedHashMap<K, String>();  
038 \* map.put(key, "A");  
039 \* map.put(key, "B");  
040 \* map.put(key, "C");  
041 \* Collection<String> coll = map.get(key);  
042 \* </pre>  
043 \* <p>  
044 \* <code>coll</code> will be a collection containing "A", "B", "C".  
045 \* </p>  
046 \*  
047 \* @param <K> the type of the keys in this map  
048 \* @param <V> the type of the values in this map  
049 \* @since 4.1  
050 \*/  
051public interface MultiValuedMap<K, V> {  
052 // Query operations  
053  
054 /\*\*  
055 \* Gets the total size of the map.  
056 \* <p>  
057 \* Implementations would return the total size of the map which is the count  
058 \* of the values from all keys.  
059 \*  
060 \* @return the total size of the map  
061 \*/  
062 int size();  
063  
064 /\*\*  
065 \* Returns {@code true} if this map contains no key-value mappings.  
066 \*  
067 \* @return {@code true} if this map contains no key-value mappings  
068 \*/  
069 boolean isEmpty();  
070  
071 /\*\*  
072 \* Returns {@code true} if this map contains a mapping for the specified  
073 \* key. More formally, returns {@code true} if and only if this map contains  
074 \* a mapping for a key {@code k} such that {@code (key==null ? k==null : key.equals(k))}.  
075 \* (There can be at most one such mapping.)  
076 \*  
077 \* @param key key whose presence in this map is to be tested  
078 \* @return true if this map contains a mapping for the specified key  
079 \* @throws NullPointerException if the specified key is null and this map  
080 \* does not permit null keys (optional)  
081 \*/  
082 boolean containsKey(Object key);  
083  
084 /\*\*  
085 \* Checks whether the map contains at least one mapping for the specified value.  
086 \*  
087 \* @param value the value to search for  
088 \* @return true if the map contains the value  
089 \* @throws NullPointerException if the value is null and null values are not supported  
090 \* by the used collection types (optional)  
091 \*/  
092 boolean containsValue(Object value);  
093  
094 /\*\*  
095 \* Checks whether the map contains a mapping for the specified key and value.  
096 \*  
097 \* @param key the key to search for  
098 \* @param value the value to search for  
099 \* @return true if the map contains the value  
100 \*/  
101 boolean containsMapping(Object key, Object value);  
102  
103 /\*\*  
104 \* Returns a view collection of the values associated with the specified key.  
105 \* <p>  
106 \* This method will return an <b>empty</b> collection if {@link #containsKey(Object)}  
107 \* returns {@code false}. Changes to the returned collection will update the underlying  
108 \* {@code MultiValuedMap} and vice-versa.  
109 \*  
110 \* @param key the key to retrieve  
111 \* @return the {@code Collection} of values, implementations should  
112 \* return an empty collection for no mapping  
113 \* @throws NullPointerException if the key is null and null keys are invalid (optional)  
114 \*/  
115 Collection<V> get(K key);  
116  
117 // Modification operations  
118  
119 /\*\*  
120 \* Adds a key-value mapping to this multi-valued map.  
121 \* <p>  
122 \* Unlike a normal {@code Map} the previous value is not replaced.  
123 \* Instead the new value is added to the collection stored against the key.  
124 \* Depending on the collection type used, duplicate key-value mappings may  
125 \* be allowed.  
126 \* <p>  
127 \* The method will return {@code true} if the size of the multi-valued map  
128 \* has been increased because of this operation.  
129 \*  
130 \* @param key the key to store against  
131 \* @param value the value to add to the collection at the key  
132 \* @return true if the map changed as a result of this put operation, or false  
133 \* if the map already contained the key-value mapping and the collection  
134 \* type does not allow duplicate values, e.g. when using a Set  
135 \* @throws UnsupportedOperationException if the put operation is not supported by  
136 \* this multi-valued map, e.g. if it is unmodifiable  
137 \* @throws NullPointerException if the key or value is null and null is invalid (optional)  
138 \* @throws IllegalArgumentException if some aspect of the specified key or value prevents  
139 \* it from being stored in this multi-valued map  
140 \*/  
141 boolean put(K key, V value);  
142  
143 /\*\*  
144 \* Adds a mapping to the specified key for all values contained in the given Iterable.  
145 \*  
146 \* @param key the key to store against  
147 \* @param values the values to add to the collection at the key, may not be null  
148 \* @return true if the map changed as a result of this operation  
149 \* @throws NullPointerException if the specified iterable is null, or if this map  
150 \* does not permit null keys or values, and the specified key or values contain  
151 \* null (optional)  
152 \*/  
153 boolean putAll(K key, Iterable<? extends V> values);  
154  
155 /\*\*  
156 \* Copies all mappings from the specified map to this multi-valued map  
157 \* (optional operation).  
158 \* <p>  
159 \* The effect of this call is equivalent to that of calling  
160 \* {@link #put(Object,Object) put(k, v)} on this map once for each mapping  
161 \* from key {@code k} to value {@code v} in the specified map.  
162 \* <p>  
163 \* The behavior of this operation is undefined if the specified map is modified  
164 \* while the operation is in progress.  
165 \*  
166 \* @param map mappings to be stored in this map, may not be null  
167 \* @return true if the map changed as a result of this operation  
168 \* @throws UnsupportedOperationException if the {@code putAll} operation is  
169 \* not supported by this map  
170 \* @throws NullPointerException if the specified map is null, or if this map  
171 \* does not permit null keys or values, and the specified map  
172 \* contains null keys or values (optional)  
173 \* @throws IllegalArgumentException if some property of a key or value in  
174 \* the specified map prevents it from being stored in this map  
175 \*/  
176 boolean putAll(Map<? extends K, ? extends V> map);  
177  
178 /\*\*  
179 \* Copies all mappings from the specified map to this multi-valued map  
180 \* (optional operation).  
181 \* <p>  
182 \* The effect of this call is equivalent to that of calling  
183 \* {@link #put(Object,Object) put(k, v)} on this map once for each  
184 \* mapping from key {@code k} to value {@code v} in the specified map.  
185 \* <p>  
186 \* The behavior of this operation is undefined if the specified map is modified  
187 \* while the operation is in progress.  
188 \*  
189 \* @param map mappings to be stored in this map, may not be null  
190 \* @return true if the map changed as a result of this operation  
191 \* @throws UnsupportedOperationException if the {@code putAll} operation is  
192 \* not supported by this map  
193 \* @throws NullPointerException if the specified map is null, or if this map  
194 \* does not permit null keys or values, and the specified map  
195 \* contains null keys or values (optional)  
196 \* @throws IllegalArgumentException if some property of a key or value in  
197 \* the specified map prevents it from being stored in this map  
198 \*/  
199 boolean putAll(MultiValuedMap<? extends K, ? extends V> map);  
200  
201 /\*\*  
202 \* Removes all values associated with the specified key.  
203 \* <p>  
204 \* The returned collection <i>may</i> be modifiable, but updates will not be propagated  
205 \* to this multi-valued map. In case no mapping was stored for the specified  
206 \* key, an empty, unmodifiable collection will be returned.  
207 \*  
208 \* @param key the key to remove values from  
209 \* @return the values that were removed  
210 \* @throws UnsupportedOperationException if the map is unmodifiable  
211 \* @throws NullPointerException if the key is null and null keys are invalid (optional)  
212 \*/  
213 Collection<V> remove(Object key);  
214  
215 /\*\*  
216 \* Removes a key-value mapping from the map.  
217 \* <p>  
218 \* The item is removed from the collection mapped to the specified key.  
219 \* Other values attached to that key are unaffected.  
220 \* <p>  
221 \* If the last value for a key is removed, implementations typically return  
222 \* an empty collection from a subsequent <code>get(Object)</code>.  
223 \*  
224 \* @param key the key to remove from  
225 \* @param item the item to remove  
226 \* @return true if the mapping was removed, false otherwise  
227 \* @throws UnsupportedOperationException if the map is unmodifiable  
228 \* @throws NullPointerException if the key or value is null and null is invalid (optional)  
229 \*/  
230 boolean removeMapping(Object key, Object item);  
231  
232 /\*\*  
233 \* Removes all of the mappings from this map (optional operation).  
234 \* <p>  
235 \* The map will be empty after this call returns.  
236 \*  
237 \* @throws UnsupportedOperationException if the map is unmodifiable  
238 \*/  
239 void clear();  
240  
241 // Views  
242  
243 /\*\*  
244 \* Returns a {@link Collection} view of the mappings contained in this multi-valued map.  
245 \* <p>  
246 \* The collection is backed by the map, so changes to the map are reflected  
247 \* in the collection, and vice-versa.  
248 \*  
249 \* @return a set view of the mappings contained in this map  
250 \*/  
251 Collection<Entry<K, V>> entries();  
252  
253 /\*\*  
254 \* Returns a {@link MultiSet} view of the keys contained in this multi-valued map.  
255 \* <p>  
256 \* The {@link MultiSet#getCount(Object)} method of the returned multiset will give  
257 \* the same result a calling {@code get(Object).size()} for the same key.  
258 \* <p>  
259 \* This multiset is backed by the map, so any changes in the map are reflected in  
260 \* the multiset.  
261 \*  
262 \* @return a multiset view of the keys contained in this map  
263 \*/  
264 MultiSet<K> keys();  
265  
266 /\*\*  
267 \* Returns a {@link Set} view of the keys contained in this multi-valued map.  
268 \* <p>  
269 \* The set is backed by the map, so changes to the map are reflected  
270 \* in the set, and vice-versa.  
271 \* <p>  
272 \* If the map is modified while an iteration over the set is in  
273 \* progress (except through the iterator's own {@code remove} operation),  
274 \* the result of the iteration is undefined. The set supports element  
275 \* removal, which removes the corresponding mapping from the map, via the  
276 \* {@code Iterator.remove}, {@code Set.remove}, {@code removeAll},  
277 \* {@code retainAll}, and {@code clear} operations. It does not support  
278 \* the {@code add} or {@code addAll} operations.  
279 \*  
280 \* @return a set view of the keys contained in this map  
281 \*/  
282 Set<K> keySet();  
283  
284 /\*\*  
285 \* Gets a {@link Collection} view of all values contained in this multi-valued map.  
286 \* <p>  
287 \* Implementations typically return a collection containing the combination  
288 \* of values from all keys.  
289 \*  
290 \* @return a collection view of the values contained in this multi-valued map  
291 \*/  
292 Collection<V> values();  
293  
294 /\*\*  
295 \* Returns a view of this multi-valued map as a {@code Map} from each distinct  
296 \* key to the non-empty collection of that key's associated values.  
297 \* <p>  
298 \* Note that {@code this.asMap().get(k)} is equivalent to {@code this.get(k)}  
299 \* only when {@code k} is a key contained in the multi-valued map; otherwise it  
300 \* returns {@code null} as opposed to an empty collection.  
301 \* <p>  
302 \* Changes to the returned map or the collections that serve as its values  
303 \* will update the underlying multi-valued map, and vice versa. The map does  
304 \* not support {@code put} or {@code putAll}, nor do its entries support  
305 \* {@link java.util.Map.Entry#setValue(Object) setValue}.  
306 \*  
307 \* @return a map view of the mappings in this multi-valued map  
308 \*/  
309 Map<K, Collection<V>> asMap();  
310  
311 // Iterators  
312  
313 /\*\*  
314 \* Obtains a <code>MapIterator</code> over this multi-valued map.  
315 \* <p>  
316 \* A map iterator is an efficient way of iterating over maps. There is no  
317 \* need to access the entries collection or use {@code Map.Entry} objects.  
318 \*  
319 \* @return a map iterator  
320 \*/  
321 MapIterator<K, V> mapIterator();  
322  
323}