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.Iterator;  
021import java.util.Set;  
022  
023/\*\*  
024 \* Defines a collection that counts the number of times an object appears in  
025 \* the collection.  
026 \* <p>  
027 \* Suppose you have a MultiSet that contains <code>{a, a, b, c}</code>.  
028 \* Calling {@link #getCount(Object)} on <code>a</code> would return 2, while  
029 \* calling {@link #uniqueSet()} would return <code>{a, b, c}</code>.  
030 \* </p>  
031 \*  
032 \* @param <E> the type held in the multiset  
033 \* @since 4.1  
034 \*/  
035public interface MultiSet<E> extends Collection<E> {  
036  
037 /\*\*  
038 \* Returns the number of occurrences of the given object currently  
039 \* in the MultiSet. If the object does not exist in the multiset,  
040 \* return 0.  
041 \*  
042 \* @param object the object to search for  
043 \* @return the number of occurrences of the object, zero if not found  
044 \*/  
045 int getCount(Object object);  
046  
047 /\*\*  
048 \* Sets the number of occurrences of the specified object in the MultiSet  
049 \* to the given count.  
050 \* <p>  
051 \* If the provided count is zero, the object will be removed from the  
052 \* {@link #uniqueSet()}.  
053 \*  
054 \* @param object the object to update  
055 \* @param count the number of occurrences of the object  
056 \* @return the number of occurrences of the object before this operation, zero  
057 \* if the object was not contained in the multiset  
058 \* @throws IllegalArgumentException if count is negative  
059 \*/  
060 int setCount(E object, int count);  
061  
062 /\*\*  
063 \* Adds one copy of the specified object to the MultiSet.  
064 \* <p>  
065 \* If the object is already in the {@link #uniqueSet()} then increment its  
066 \* count as reported by {@link #getCount(Object)}. Otherwise add it to the  
067 \* {@link #uniqueSet()} and report its count as 1.  
068 \*  
069 \* @param object the object to add  
070 \* @return <code>true</code> always, as the size of the MultiSet is increased  
071 \* in any case  
072 \*/  
073 @Override  
074 boolean add(E object);  
075  
076 /\*\*  
077 \* Adds a number of occurrences of the specified object to the MultiSet.  
078 \* <p>  
079 \* If the object is already in the {@link #uniqueSet()} then increment its  
080 \* count as reported by {@link #getCount(Object)}. Otherwise add it to the  
081 \* {@link #uniqueSet()} and report its count as <code>occurrences</code>.  
082 \*  
083 \* @param object the object to add  
084 \* @param occurrences the number of occurrences to add, may be zero,  
085 \* in which case no change is made to the multiset  
086 \* @return the number of occurrences of the object in the multiset before  
087 \* this operation; possibly zero  
088 \* @throws IllegalArgumentException if occurrences is negative  
089 \*/  
090 int add(E object, int occurrences);  
091  
092 /\*\*  
093 \* Removes one occurrence of the given object from the MultiSet.  
094 \* <p>  
095 \* If the number of occurrences after this operations is reduced  
096 \* to zero, the object will be removed from the {@link #uniqueSet()}.  
097 \*  
098 \* @param object the object to remove  
099 \* @return <code>true</code> if this call changed the collection  
100 \*/  
101 @Override  
102 boolean remove(Object object);  
103  
104 /\*\*  
105 \* Removes a number of occurrences of the specified object from the MultiSet.  
106 \* <p>  
107 \* If the number of occurrences to remove is greater than the actual number of  
108 \* occurrences in the multiset, the object will be removed from the multiset.  
109 \*  
110 \* @param object the object to remove  
111 \* @param occurrences the number of occurrences to remove, may be zero,  
112 \* in which case no change is made to the multiset  
113 \* @return the number of occurrences of the object in the multiset  
114 \* before the operation; possibly zero  
115 \* @throws IllegalArgumentException if occurrences is negative  
116 \*/  
117 int remove(Object object, int occurrences);  
118  
119 /\*\*  
120 \* Returns a {@link Set} of unique elements in the MultiSet.  
121 \* <p>  
122 \* Uniqueness constraints are the same as those in {@link java.util.Set}.  
123 \* <p>  
124 \* The returned set is backed by this multiset, so any change to either  
125 \* is immediately reflected in the other. Only removal operations are  
126 \* supported, in which case all occurrences of the element are removed  
127 \* from the backing multiset.  
128 \*  
129 \* @return the Set of unique MultiSet elements  
130 \*/  
131 Set<E> uniqueSet();  
132  
133 /\*\*  
134 \* Returns a {@link Set} of all entries contained in the MultiSet.  
135 \* <p>  
136 \* The returned set is backed by this multiset, so any change to either  
137 \* is immediately reflected in the other.  
138 \*  
139 \* @return the Set of MultiSet entries  
140 \*/  
141 Set<Entry<E>> entrySet();  
142  
143 /\*\*  
144 \* Returns an {@link Iterator} over the entire set of members,  
145 \* including copies due to cardinality. This iterator is fail-fast  
146 \* and will not tolerate concurrent modifications.  
147 \*  
148 \* @return iterator over all elements in the MultiSet  
149 \*/  
150 @Override  
151 Iterator<E> iterator();  
152  
153 /\*\*  
154 \* Returns the total number of items in the MultiSet.  
155 \*  
156 \* @return the total size of the multiset  
157 \*/  
158 @Override  
159 int size();  
160  
161 /\*\*  
162 \* Returns <code>true</code> if the MultiSet contains at least one  
163 \* occurrence for each element contained in the given collection.  
164 \*  
165 \* @param coll the collection to check against  
166 \* @return <code>true</code> if the MultiSet contains all the collection  
167 \*/  
168 @Override  
169 boolean containsAll(Collection<?> coll);  
170  
171 /\*\*  
172 \* Remove all occurrences of all elements from this MultiSet represented  
173 \* in the given collection.  
174 \*  
175 \* @param coll the collection of elements to remove  
176 \* @return <code>true</code> if this call changed the multiset  
177 \*/  
178 @Override  
179 boolean removeAll(Collection<?> coll);  
180  
181 /\*\*  
182 \* Remove any elements of this MultiSet that are not contained in the  
183 \* given collection.  
184 \*  
185 \* @param coll the collection of elements to retain  
186 \* @return <code>true</code> if this call changed the multiset  
187 \*/  
188 @Override  
189 boolean retainAll(Collection<?> coll);  
190  
191 /\*\*  
192 \* Compares this MultiSet to another object.  
193 \* <p>  
194 \* This MultiSet equals another object if it is also a MultiSet  
195 \* that contains the same number of occurrences of the same elements.  
196 \*  
197 \* @param obj the object to compare to  
198 \* @return true if equal  
199 \*/  
200 @Override  
201 boolean equals(Object obj);  
202  
203 /\*\*  
204 \* Gets a hash code for the MultiSet compatible with the definition of equals.  
205 \* The hash code is defined as the sum total of a hash code for each element.  
206 \* The per element hash code is defined as  
207 \* <code>(e==null ? 0 : e.hashCode()) ^ noOccurances)</code>.  
208 \*  
209 \* @return the hash code of the MultiSet  
210 \*/  
211 @Override  
212 int hashCode();  
213  
214 /\*\*  
215 \* An unmodifiable entry for an element and its occurrence as contained in a MultiSet.  
216 \* <p>  
217 \* The {@link MultiSet#entrySet()} method returns a view of the multiset whose elements  
218 \* implements this interface.  
219 \*  
220 \* @param <E> the element type  
221 \*/  
222 interface Entry<E> {  
223  
224 /\*\*  
225 \* Returns the element corresponding to this entry.  
226 \*  
227 \* @return the element corresponding to this entry  
228 \*/  
229 E getElement();  
230  
231 /\*\*  
232 \* Returns the number of occurrences for the element of this entry.  
233 \*  
234 \* @return the number of occurrences of the element  
235 \*/  
236 int getCount();  
237  
238 /\*\*  
239 \* Compares the specified object with this entry for equality.  
240 \* Returns true if the given object is also a multiset entry  
241 \* and the two entries represent the same element with the same  
242 \* number of occurrences.  
243 \* <p>  
244 \* More formally, two entries <code>e1</code> and <code>e2</code> represent  
245 \* the same mapping if  
246 \* <pre>  
247 \* (e1.getElement()==null ? e2.getElement()==null  
248 \* : e1.getElement().equals(e2.getElement())) &&  
249 \* (e1.getCount()==e2.getCount())  
250 \* </pre>  
251 \*  
252 \* @param o object to be compared for equality with this multiset entry  
253 \* @return true if the specified object is equal to this multiset entry  
254 \*/  
255 @Override  
256 boolean equals(Object o);  
257  
258 /\*\*  
259 \* Returns the hash code value for this multiset entry.  
260 \* <p>  
261 \* The hash code of a multiset entry <code>e</code> is defined to be:  
262 \* <pre>  
263 \* (e==null ? 0 : e.hashCode()) ^ noOccurances)  
264 \* </pre>  
265 \*  
266 \* @return the hash code value for this multiset entry  
267 \*/  
268 @Override  
269 int hashCode();  
270 }  
271  
272}