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 Bag 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 \* <p>  
032 \* <i>NOTE: This interface violates the {@link Collection} contract.</i>  
033 \* The behavior specified in many of these methods is <i>not</i> the same  
034 \* as the behavior specified by <code>Collection</code>.  
035 \* The noncompliant methods are clearly marked with "(Violation)".  
036 \* Exercise caution when using a bag as a <code>Collection</code>.  
037 \* </p>  
038 \* <p>  
039 \* This violation resulted from the original specification of this interface.  
040 \* In an ideal world, the interface would be changed to fix the problems, however  
041 \* it has been decided to maintain backwards compatibility instead.  
042 \* </p>  
043 \*  
044 \* @param <E> the type of elements in this bag  
045 \* @since 2.0  
046 \*/  
047public interface Bag<E> extends Collection<E> {  
048  
049 /\*\*  
050 \* Returns the number of occurrences (cardinality) of the given  
051 \* object currently in the bag. If the object does not exist in the  
052 \* bag, return 0.  
053 \*  
054 \* @param object the object to search for  
055 \* @return the number of occurrences of the object, zero if not found  
056 \*/  
057 int getCount(Object object);  
058  
059 /\*\*  
060 \* <i>(Violation)</i>  
061 \* Adds one copy of the specified object to the Bag.  
062 \* <p>  
063 \* If the object is already in the {@link #uniqueSet()} then increment its  
064 \* count as reported by {@link #getCount(Object)}. Otherwise add it to the  
065 \* {@link #uniqueSet()} and report its count as 1.  
066 \* </p>  
067 \* <p>  
068 \* Since this method always increases the size of the bag,  
069 \* according to the {@link Collection#add(Object)} contract, it  
070 \* should always return <code>true</code>. Since it sometimes returns  
071 \* <code>false</code>, this method violates the contract.  
072 \* </p>  
073 \*  
074 \* @param object the object to add  
075 \* @return <code>true</code> if the object was not already in the <code>uniqueSet</code>  
076 \*/  
077 @Override  
078 boolean add(E object);  
079  
080 /\*\*  
081 \* Adds <code>nCopies</code> copies of the specified object to the Bag.  
082 \* <p>  
083 \* If the object is already in the {@link #uniqueSet()} then increment its  
084 \* count as reported by {@link #getCount(Object)}. Otherwise add it to the  
085 \* {@link #uniqueSet()} and report its count as <code>nCopies</code>.  
086 \* </p>  
087 \*  
088 \* @param object the object to add  
089 \* @param nCopies the number of copies to add  
090 \* @return <code>true</code> if the object was not already in the <code>uniqueSet</code>  
091 \*/  
092 boolean add(E object, int nCopies);  
093  
094 /\*\*  
095 \* <i>(Violation)</i>  
096 \* Removes all occurrences of the given object from the bag.  
097 \* <p>  
098 \* This will also remove the object from the {@link #uniqueSet()}.  
099 \* </p>  
100 \* <p>  
101 \* According to the {@link Collection#remove(Object)} method,  
102 \* this method should only remove the <i>first</i> occurrence of the  
103 \* given object, not <i>all</i> occurrences.  
104 \* </p>  
105 \*  
106 \* @param object the object to remove  
107 \* @return <code>true</code> if this call changed the collection  
108 \*/  
109 @Override  
110 boolean remove(Object object);  
111  
112 /\*\*  
113 \* Removes <code>nCopies</code> copies of the specified object from the Bag.  
114 \* <p>  
115 \* If the number of copies to remove is greater than the actual number of  
116 \* copies in the Bag, no error is thrown.  
117 \* </p>  
118 \*  
119 \* @param object the object to remove  
120 \* @param nCopies the number of copies to remove  
121 \* @return <code>true</code> if this call changed the collection  
122 \*/  
123 boolean remove(Object object, int nCopies);  
124  
125 /\*\*  
126 \* Returns a {@link Set} of unique elements in the Bag.  
127 \* <p>  
128 \* Uniqueness constraints are the same as those in {@link java.util.Set}.  
129 \* </p>  
130 \*  
131 \* @return the Set of unique Bag elements  
132 \*/  
133 Set<E> uniqueSet();  
134  
135 /\*\*  
136 \* Returns the total number of items in the bag across all types.  
137 \*  
138 \* @return the total size of the Bag  
139 \*/  
140 @Override  
141 int size();  
142  
143 /\*\*  
144 \* <i>(Violation)</i>  
145 \* Returns <code>true</code> if the bag contains all elements in  
146 \* the given collection, respecting cardinality. That is, if the  
147 \* given collection <code>coll</code> contains <code>n</code> copies  
148 \* of a given object, calling {@link #getCount(Object)} on that object must  
149 \* be <code>>= n</code> for all <code>n</code> in <code>coll</code>.  
150 \*  
151 \* <p>  
152 \* The {@link Collection#containsAll(Collection)} method specifies  
153 \* that cardinality should <i>not</i> be respected; this method should  
154 \* return true if the bag contains at least one of every object contained  
155 \* in the given collection.  
156 \* </p>  
157 \*  
158 \* @param coll the collection to check against  
159 \* @return <code>true</code> if the Bag contains all the collection  
160 \*/  
161 @Override  
162 boolean containsAll(Collection<?> coll);  
163  
164 /\*\*  
165 \* <i>(Violation)</i>  
166 \* Remove all elements represented in the given collection,  
167 \* respecting cardinality. That is, if the given collection  
168 \* <code>coll</code> contains <code>n</code> copies of a given object,  
169 \* the bag will have <code>n</code> fewer copies, assuming the bag  
170 \* had at least <code>n</code> copies to begin with.  
171 \*  
172 \* <p>  
173 \* The {@link Collection#removeAll(Collection)} method specifies  
174 \* that cardinality should <i>not</i> be respected; this method should  
175 \* remove <i>all</i> occurrences of every object contained in the  
176 \* given collection.  
177 \* </p>  
178 \*  
179 \* @param coll the collection to remove  
180 \* @return <code>true</code> if this call changed the collection  
181 \*/  
182 @Override  
183 boolean removeAll(Collection<?> coll);  
184  
185 /\*\*  
186 \* <i>(Violation)</i>  
187 \* Remove any members of the bag that are not in the given  
188 \* collection, respecting cardinality. That is, if the given  
189 \* collection <code>coll</code> contains <code>n</code> copies of a  
190 \* given object and the bag has <code>m > n</code> copies, then  
191 \* delete <code>m - n</code> copies from the bag. In addition, if  
192 \* <code>e</code> is an object in the bag but  
193 \* <code>!coll.contains(e)</code>, then remove <code>e</code> and any  
194 \* of its copies.  
195 \*  
196 \* <p>  
197 \* The {@link Collection#retainAll(Collection)} method specifies  
198 \* that cardinality should <i>not</i> be respected; this method should  
199 \* keep <i>all</i> occurrences of every object contained in the  
200 \* given collection.  
201 \* </p>  
202 \*  
203 \* @param coll the collection to retain  
204 \* @return <code>true</code> if this call changed the collection  
205 \*/  
206 @Override  
207 boolean retainAll(Collection<?> coll);  
208  
209 /\*\*  
210 \* Returns an {@link Iterator} over the entire set of members,  
211 \* including copies due to cardinality. This iterator is fail-fast  
212 \* and will not tolerate concurrent modifications.  
213 \*  
214 \* @return iterator over all elements in the Bag  
215 \*/  
216 @Override  
217 Iterator<E> iterator();  
218  
219 // The following is not part of the formal Bag interface, however where possible  
220 // Bag implementations should follow these comments.  
221// /\*\*  
222// \* Compares this Bag to another.  
223// \* This Bag equals another Bag if it contains the same number of occurrences of  
224// \* the same elements.  
225// \* This equals definition is compatible with the Set interface.  
226// \*  
227// \* @param obj the Bag to compare to  
228// \* @return true if equal  
229// \*/  
230// boolean equals(Object obj);  
231//  
232// /\*\*  
233// \* Gets a hash code for the Bag compatible with the definition of equals.  
234// \* The hash code is defined as the sum total of a hash code for each element.  
235// \* The per element hash code is defined as  
236// \* <code>(e==null ? 0 : e.hashCode()) ^ noOccurances)</code>.  
237// \* This hash code definition is compatible with the Set interface.  
238// \*  
239// \* @return the hash code of the Bag  
240// \*/  
241// int hashCode();  
242  
243}