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.bag;  
018  
019import java.util.Set;  
020  
021import org.apache.commons.collections4.Bag;  
022import org.apache.commons.collections4.Transformer;  
023import org.apache.commons.collections4.collection.TransformedCollection;  
024import org.apache.commons.collections4.set.TransformedSet;  
025  
026/\*\*  
027 \* Decorates another {@link Bag} to transform objects that are added.  
028 \* <p>  
029 \* The add methods are affected by this class.  
030 \* Thus objects must be removed or searched for using their transformed form.  
031 \* For example, if the transformation converts Strings to Integers, you must  
032 \* use the Integer form to remove objects.  
033 \* </p>  
034 \* <p>  
035 \* This class is Serializable from Commons Collections 3.1.  
036 \* </p>  
037 \*  
038 \* @param <E> the type of elements in this bag  
039 \* @since 3.0  
040 \*/  
041public class TransformedBag<E> extends TransformedCollection<E> implements Bag<E> {  
042  
043 /\*\* Serialization version \*/  
044 private static final long serialVersionUID = 5421170911299074185L;  
045  
046 /\*\*  
047 \* Factory method to create a transforming bag.  
048 \* <p>  
049 \* If there are any elements already in the bag being decorated, they  
050 \* are NOT transformed. Contrast this with {@link #transformedBag(Bag, Transformer)}.  
051 \*  
052 \* @param <E> the type of the elements in the bag  
053 \* @param bag the bag to decorate, must not be null  
054 \* @param transformer the transformer to use for conversion, must not be null  
055 \* @return a new transformed Bag  
056 \* @throws NullPointerException if bag or transformer is null  
057 \* @since 4.0  
058 \*/  
059 public static <E> Bag<E> transformingBag(final Bag<E> bag, final Transformer<? super E, ? extends E> transformer) {  
060 return new TransformedBag<>(bag, transformer);  
061 }  
062  
063 /\*\*  
064 \* Factory method to create a transforming bag that will transform  
065 \* existing contents of the specified bag.  
066 \* <p>  
067 \* If there are any elements already in the bag being decorated, they  
068 \* will be transformed by this method.  
069 \* Contrast this with {@link #transformingBag(Bag, Transformer)}.  
070 \*  
071 \* @param <E> the type of the elements in the bag  
072 \* @param bag the bag to decorate, must not be null  
073 \* @param transformer the transformer to use for conversion, must not be null  
074 \* @return a new transformed Bag  
075 \* @throws NullPointerException if bag or transformer is null  
076 \* @since 4.0  
077 \*/  
078 public static <E> Bag<E> transformedBag(final Bag<E> bag, final Transformer<? super E, ? extends E> transformer) {  
079 final TransformedBag<E> decorated = new TransformedBag<>(bag, transformer);  
080 if (bag.size() > 0) {  
081 @SuppressWarnings("unchecked") // Bag is of type E  
082 final E[] values = (E[]) bag.toArray(); // NOPMD - false positive for generics  
083 bag.clear();  
084 for (final E value : values) {  
085 decorated.decorated().add(transformer.transform(value));  
086 }  
087 }  
088 return decorated;  
089 }  
090  
091 //-----------------------------------------------------------------------  
092 /\*\*  
093 \* Constructor that wraps (not copies).  
094 \* <p>  
095 \* If there are any elements already in the bag being decorated, they  
096 \* are NOT transformed.  
097 \*  
098 \* @param bag the bag to decorate, must not be null  
099 \* @param transformer the transformer to use for conversion, must not be null  
100 \* @throws NullPointerException if bag or transformer is null  
101 \*/  
102 protected TransformedBag(final Bag<E> bag, final Transformer<? super E, ? extends E> transformer) {  
103 super(bag, transformer);  
104 }  
105  
106 /\*\*  
107 \* Gets the decorated bag.  
108 \*  
109 \* @return the decorated bag  
110 \*/  
111 protected Bag<E> getBag() {  
112 return (Bag<E>) decorated();  
113 }  
114  
115 @Override  
116 public boolean equals(final Object object) {  
117 return object == this || decorated().equals(object);  
118 }  
119  
120 @Override  
121 public int hashCode() {  
122 return decorated().hashCode();  
123 }  
124  
125 //-----------------------------------------------------------------------  
126  
127 @Override  
128 public int getCount(final Object object) {  
129 return getBag().getCount(object);  
130 }  
131  
132 @Override  
133 public boolean remove(final Object object, final int nCopies) {  
134 return getBag().remove(object, nCopies);  
135 }  
136  
137 //-----------------------------------------------------------------------  
138  
139 @Override  
140 public boolean add(final E object, final int nCopies) {  
141 return getBag().add(transform(object), nCopies);  
142 }  
143  
144 @Override  
145 public Set<E> uniqueSet() {  
146 final Set<E> set = getBag().uniqueSet();  
147 return TransformedSet.<E>transformingSet(set, transformer);  
148 }  
149  
150}