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.Comparator;  
020  
021import org.apache.commons.collections4.SortedBag;  
022import org.apache.commons.collections4.Transformer;  
023  
024/\*\*  
025 \* Decorates another {@link SortedBag} to transform objects that are added.  
026 \* <p>  
027 \* The add methods are affected by this class.  
028 \* Thus objects must be removed or searched for using their transformed form.  
029 \* For example, if the transformation converts Strings to Integers, you must  
030 \* use the Integer form to remove objects.  
031 \* </p>  
032 \* <p>  
033 \* This class is Serializable from Commons Collections 3.1.  
034 \* </p>  
035 \*  
036 \* @param <E> the type of elements in this bag  
037 \* @since 3.0  
038 \*/  
039public class TransformedSortedBag<E> extends TransformedBag<E> implements SortedBag<E> {  
040  
041 /\*\* Serialization version \*/  
042 private static final long serialVersionUID = -251737742649401930L;  
043  
044 /\*\*  
045 \* Factory method to create a transforming sorted bag.  
046 \* <p>  
047 \* If there are any elements already in the bag being decorated, they  
048 \* are NOT transformed. Contrast this with {@link #transformedSortedBag(SortedBag, Transformer)}.  
049 \*  
050 \* @param <E> the type of the elements in the bag  
051 \* @param bag the bag to decorate, must not be null  
052 \* @param transformer the transformer to use for conversion, must not be null  
053 \* @return a new transformed SortedBag  
054 \* @throws NullPointerException if bag or transformer is null  
055 \* @since 4.0  
056 \*/  
057 public static <E> TransformedSortedBag<E> transformingSortedBag(final SortedBag<E> bag,  
058 final Transformer<? super E, ? extends E> transformer) {  
059 return new TransformedSortedBag<>(bag, transformer);  
060 }  
061  
062 /\*\*  
063 \* Factory method to create a transforming sorted bag that will transform  
064 \* existing contents of the specified sorted bag.  
065 \* <p>  
066 \* If there are any elements already in the bag being decorated, they  
067 \* will be transformed by this method.  
068 \* Contrast this with {@link #transformingSortedBag(SortedBag, Transformer)}.  
069 \*  
070 \* @param <E> the type of the elements in the bag  
071 \* @param bag the bag to decorate, must not be null  
072 \* @param transformer the transformer to use for conversion, must not be null  
073 \* @return a new transformed SortedBag  
074 \* @throws NullPointerException if bag or transformer is null  
075 \* @since 4.0  
076 \*/  
077 public static <E> TransformedSortedBag<E> transformedSortedBag(final SortedBag<E> bag,  
078 final Transformer<? super E, ? extends E> transformer) {  
079  
080 final TransformedSortedBag<E> decorated = new TransformedSortedBag<>(bag, transformer);  
081 if (bag.size() > 0) {  
082 @SuppressWarnings("unchecked") // bag is type E  
083 final E[] values = (E[]) bag.toArray(); // NOPMD - false positive for generics  
084 bag.clear();  
085 for (final E value : values) {  
086 decorated.decorated().add(transformer.transform(value));  
087 }  
088 }  
089 return decorated;  
090 }  
091  
092 //-----------------------------------------------------------------------  
093 /\*\*  
094 \* Constructor that wraps (not copies).  
095 \* <p>  
096 \* If there are any elements already in the bag being decorated, they  
097 \* are NOT transformed.  
098 \*  
099 \* @param bag the bag to decorate, must not be null  
100 \* @param transformer the transformer to use for conversion, must not be null  
101 \* @throws NullPointerException if bag or transformer is null  
102 \*/  
103 protected TransformedSortedBag(final SortedBag<E> bag, final Transformer<? super E, ? extends E> transformer) {  
104 super(bag, transformer);  
105 }  
106  
107 /\*\*  
108 \* Gets the decorated bag.  
109 \*  
110 \* @return the decorated bag  
111 \*/  
112 protected SortedBag<E> getSortedBag() {  
113 return (SortedBag<E>) decorated();  
114 }  
115  
116 //-----------------------------------------------------------------------  
117  
118 @Override  
119 public E first() {  
120 return getSortedBag().first();  
121 }  
122  
123 @Override  
124 public E last() {  
125 return getSortedBag().last();  
126 }  
127  
128 @Override  
129 public Comparator<? super E> comparator() {  
130 return getSortedBag().comparator();  
131 }  
132  
133}