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.set;  
018  
019import java.util.Iterator;  
020import java.util.NavigableSet;  
021  
022import org.apache.commons.collections4.Transformer;  
023  
024/\*\*  
025 \* Decorates another <code>NavigableSet</code> 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 \*  
033 \* @param <E> the type of the elements in this set  
034 \* @since 4.1  
035 \*/  
036public class TransformedNavigableSet<E> extends TransformedSortedSet<E> implements NavigableSet<E> {  
037  
038 /\*\* Serialization version \*/  
039 private static final long serialVersionUID = 20150528L;  
040  
041 /\*\*  
042 \* Factory method to create a transforming navigable set.  
043 \* <p>  
044 \* If there are any elements already in the set being decorated, they  
045 \* are NOT transformed.  
046 \* Contrast this with {@link #transformedNavigableSet(NavigableSet, Transformer)}.  
047 \*  
048 \* @param <E> the element type  
049 \* @param set the set to decorate, must not be null  
050 \* @param transformer the transformer to use for conversion, must not be null  
051 \* @return a new transformed {@link NavigableSet}  
052 \* @throws NullPointerException if set or transformer is null  
053 \*/  
054 public static <E> TransformedNavigableSet<E> transformingNavigableSet(final NavigableSet<E> set,  
055 final Transformer<? super E, ? extends E> transformer) {  
056 return new TransformedNavigableSet<>(set, transformer);  
057 }  
058  
059 /\*\*  
060 \* Factory method to create a transforming navigable set that will transform  
061 \* existing contents of the specified navigable set.  
062 \* <p>  
063 \* If there are any elements already in the set being decorated, they  
064 \* will be transformed by this method.  
065 \* Contrast this with {@link #transformingNavigableSet(NavigableSet, Transformer)}.  
066 \*  
067 \* @param <E> the element type  
068 \* @param set the set to decorate, must not be null  
069 \* @param transformer the transformer to use for conversion, must not be null  
070 \* @return a new transformed {@link NavigableSet}  
071 \* @throws NullPointerException if set or transformer is null  
072 \*/  
073 public static <E> TransformedNavigableSet<E> transformedNavigableSet(final NavigableSet<E> set,  
074 final Transformer<? super E, ? extends E> transformer) {  
075  
076 final TransformedNavigableSet<E> decorated = new TransformedNavigableSet<>(set, transformer);  
077 if (set.size() > 0) {  
078 @SuppressWarnings("unchecked") // set is type E  
079 final E[] values = (E[]) set.toArray(); // NOPMD - false positive for generics  
080 set.clear();  
081 for (final E value : values) {  
082 decorated.decorated().add(transformer.transform(value));  
083 }  
084 }  
085 return decorated;  
086 }  
087  
088 //-----------------------------------------------------------------------  
089 /\*\*  
090 \* Constructor that wraps (not copies).  
091 \* <p>  
092 \* If there are any elements already in the set being decorated, they  
093 \* are NOT transformed.  
094 \*  
095 \* @param set the set to decorate, must not be null  
096 \* @param transformer the transformer to use for conversion, must not be null  
097 \* @throws NullPointerException if set or transformer is null  
098 \*/  
099 protected TransformedNavigableSet(final NavigableSet<E> set,  
100 final Transformer<? super E, ? extends E> transformer) {  
101 super(set, transformer);  
102 }  
103  
104 /\*\*  
105 \* Gets the decorated navigable set.  
106 \*  
107 \* @return the decorated navigable set  
108 \*/  
109 @Override  
110 protected NavigableSet<E> decorated() {  
111 return (NavigableSet<E>) super.decorated();  
112 }  
113  
114 //-----------------------------------------------------------------------  
115  
116 @Override  
117 public E lower(final E e) {  
118 return decorated().lower(e);  
119 }  
120  
121 @Override  
122 public E floor(final E e) {  
123 return decorated().floor(e);  
124 }  
125  
126 @Override  
127 public E ceiling(final E e) {  
128 return decorated().ceiling(e);  
129 }  
130  
131 @Override  
132 public E higher(final E e) {  
133 return decorated().higher(e);  
134 }  
135  
136 @Override  
137 public E pollFirst() {  
138 return decorated().pollFirst();  
139 }  
140  
141 @Override  
142 public E pollLast() {  
143 return decorated().pollLast();  
144 }  
145  
146 @Override  
147 public NavigableSet<E> descendingSet() {  
148 return transformingNavigableSet(decorated().descendingSet(), transformer);  
149 }  
150  
151 @Override  
152 public Iterator<E> descendingIterator() {  
153 return decorated().descendingIterator();  
154 }  
155  
156 @Override  
157 public NavigableSet<E> subSet(final E fromElement, final boolean fromInclusive, final E toElement,  
158 final boolean toInclusive) {  
159 final NavigableSet<E> sub = decorated().subSet(fromElement, fromInclusive, toElement, toInclusive);  
160 return transformingNavigableSet(sub, transformer);  
161 }  
162  
163 @Override  
164 public NavigableSet<E> headSet(final E toElement, final boolean inclusive) {  
165 final NavigableSet<E> head = decorated().headSet(toElement, inclusive);  
166 return transformingNavigableSet(head, transformer);  
167 }  
168  
169 @Override  
170 public NavigableSet<E> tailSet(final E fromElement, final boolean inclusive) {  
171 final NavigableSet<E> tail = decorated().tailSet(fromElement, inclusive);  
172 return transformingNavigableSet(tail, transformer);  
173 }  
174  
175}