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  
022/\*\*  
023 \* Decorates another <code>NavigableSet</code> to provide additional behaviour.  
024 \* <p>  
025 \* Methods are forwarded directly to the decorated set.  
026 \* </p>  
027 \*  
028 \* @param <E> the type of the elements in the navigable set  
029 \* @since 4.1  
030 \*/  
031public abstract class AbstractNavigableSetDecorator<E>  
032 extends AbstractSortedSetDecorator<E>  
033 implements NavigableSet<E> {  
034  
035 /\*\* Serialization version \*/  
036 private static final long serialVersionUID = 20150528L;  
037  
038 /\*\*  
039 \* Constructor only used in deserialization, do not use otherwise.  
040 \*/  
041 protected AbstractNavigableSetDecorator() {  
042 super();  
043 }  
044  
045 /\*\*  
046 \* Constructor that wraps (not copies).  
047 \*  
048 \* @param set the set to decorate, must not be null  
049 \* @throws NullPointerException if set is null  
050 \*/  
051 protected AbstractNavigableSetDecorator(final NavigableSet<E> set) {  
052 super(set);  
053 }  
054  
055 /\*\*  
056 \* Gets the set being decorated.  
057 \*  
058 \* @return the decorated set  
059 \*/  
060 @Override  
061 protected NavigableSet<E> decorated() {  
062 return (NavigableSet<E>) super.decorated();  
063 }  
064  
065 //-----------------------------------------------------------------------  
066  
067 @Override  
068 public E lower(final E e) {  
069 return decorated().lower(e);  
070 }  
071  
072 @Override  
073 public E floor(final E e) {  
074 return decorated().floor(e);  
075 }  
076  
077 @Override  
078 public E ceiling(final E e) {  
079 return decorated().ceiling(e);  
080 }  
081  
082 @Override  
083 public E higher(final E e) {  
084 return decorated().higher(e);  
085 }  
086  
087 @Override  
088 public E pollFirst() {  
089 return decorated().pollFirst();  
090 }  
091  
092 @Override  
093 public E pollLast() {  
094 return decorated().pollLast();  
095 }  
096  
097 @Override  
098 public NavigableSet<E> descendingSet() {  
099 return decorated().descendingSet();  
100 }  
101  
102 @Override  
103 public Iterator<E> descendingIterator() {  
104 return decorated().descendingIterator();  
105 }  
106  
107 @Override  
108 public NavigableSet<E> subSet(final E fromElement, final boolean fromInclusive, final E toElement,  
109 final boolean toInclusive) {  
110 return decorated().subSet(fromElement, fromInclusive, toElement, toInclusive);  
111 }  
112  
113 @Override  
114 public NavigableSet<E> headSet(final E toElement, final boolean inclusive) {  
115 return decorated().headSet(toElement, inclusive);  
116 }  
117  
118 @Override  
119 public NavigableSet<E> tailSet(final E fromElement, final boolean inclusive) {  
120 return decorated().tailSet(fromElement, inclusive);  
121 }  
122  
123}