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.Bag;  
022import org.apache.commons.collections4.SortedBag;  
023  
024/\*\*  
025 \* Decorates another {@link SortedBag} to synchronize its behaviour  
026 \* for a multi-threaded environment.  
027 \* <p>  
028 \* Methods are synchronized, then forwarded to the decorated bag.  
029 \* Iterators must be separately synchronized around the loop.  
030 \* </p>  
031 \* <p>  
032 \* This class is Serializable from Commons Collections 3.1.  
033 \* </p>  
034 \*  
035 \* @param <E> the type of elements in this bag  
036 \* @since 3.0  
037 \*/  
038public class SynchronizedSortedBag<E> extends SynchronizedBag<E> implements SortedBag<E> {  
039  
040 /\*\* Serialization version \*/  
041 private static final long serialVersionUID = 722374056718497858L;  
042  
043 /\*\*  
044 \* Factory method to create a synchronized sorted bag.  
045 \*  
046 \* @param <E> the type of the elements in the bag  
047 \* @param bag the bag to decorate, must not be null  
048 \* @return a new synchronized SortedBag  
049 \* @throws NullPointerException if bag is null  
050 \* @since 4.0  
051 \*/  
052 public static <E> SynchronizedSortedBag<E> synchronizedSortedBag(final SortedBag<E> bag) {  
053 return new SynchronizedSortedBag<>(bag);  
054 }  
055  
056 //-----------------------------------------------------------------------  
057 /\*\*  
058 \* Constructor that wraps (not copies).  
059 \*  
060 \* @param bag the bag to decorate, must not be null  
061 \* @throws NullPointerException if bag is null  
062 \*/  
063 protected SynchronizedSortedBag(final SortedBag<E> bag) {  
064 super(bag);  
065 }  
066  
067 /\*\*  
068 \* Constructor that wraps (not copies).  
069 \*  
070 \* @param bag the bag to decorate, must not be null  
071 \* @param lock the lock to use, must not be null  
072 \* @throws NullPointerException if bag or lock is null  
073 \*/  
074 protected SynchronizedSortedBag(final Bag<E> bag, final Object lock) {  
075 super(bag, lock);  
076 }  
077  
078 /\*\*  
079 \* Gets the bag being decorated.  
080 \*  
081 \* @return the decorated bag  
082 \*/  
083 protected SortedBag<E> getSortedBag() {  
084 return (SortedBag<E>) decorated();  
085 }  
086  
087 //-----------------------------------------------------------------------  
088  
089 @Override  
090 public synchronized E first() {  
091 synchronized (lock) {  
092 return getSortedBag().first();  
093 }  
094 }  
095  
096 @Override  
097 public synchronized E last() {  
098 synchronized (lock) {  
099 return getSortedBag().last();  
100 }  
101 }  
102  
103 @Override  
104 public synchronized Comparator<? super E> comparator() {  
105 synchronized (lock) {  
106 return getSortedBag().comparator();  
107 }  
108 }  
109  
110}