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.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.io.Serializable;  
023import java.util.Collection;  
024import java.util.Comparator;  
025import java.util.SortedMap;  
026import java.util.TreeMap;  
027  
028import org.apache.commons.collections4.SortedBag;  
029  
030/\*\*  
031 \* Implements {@link SortedBag}, using a {@link TreeMap} to provide the data storage.  
032 \* This is the standard implementation of a sorted bag.  
033 \* <p>  
034 \* Order will be maintained among the bag members and can be viewed through the iterator.  
035 \* </p>  
036 \* <p>  
037 \* A {@link org.apache.commons.collections4.Bag Bag} stores each object in the collection  
038 \* together with a count of occurrences. Extra methods on the interface allow multiple  
039 \* copies of an object to be added or removed at once. It is important to read the interface  
040 \* javadoc carefully as several methods violate the {@link Collection} interface specification.  
041 \* </p>  
042 \*  
043 \* @param <E> the type of elements in this bag  
044 \* @since 3.0 (previously in main package v2.0)  
045 \*/  
046public class TreeBag<E> extends AbstractMapBag<E> implements SortedBag<E>, Serializable {  
047  
048 /\*\* Serial version lock \*/  
049 private static final long serialVersionUID = -7740146511091606676L;  
050  
051 /\*\*  
052 \* Constructs an empty {@link TreeBag}.  
053 \*/  
054 public TreeBag() {  
055 super(new TreeMap<E, MutableInteger>());  
056 }  
057  
058 /\*\*  
059 \* Constructs an empty bag that maintains order on its unique representative  
060 \* members according to the given {@link Comparator}.  
061 \*  
062 \* @param comparator the comparator to use  
063 \*/  
064 public TreeBag(final Comparator<? super E> comparator) {  
065 super(new TreeMap<E, MutableInteger>(comparator));  
066 }  
067  
068 /\*\*  
069 \* Constructs a {@link TreeBag} containing all the members of the  
070 \* specified collection.  
071 \*  
072 \* @param coll the collection to copy into the bag  
073 \*/  
074 public TreeBag(final Collection<? extends E> coll) {  
075 this();  
076 addAll(coll);  
077 }  
078  
079 //-----------------------------------------------------------------------  
080 /\*\*  
081 \* {@inheritDoc}  
082 \*  
083 \* @throws IllegalArgumentException if the object to be added does not implement  
084 \* {@link Comparable} and the {@link TreeBag} is using natural ordering  
085 \* @throws NullPointerException if the specified key is null and this bag uses  
086 \* natural ordering, or its comparator does not permit null keys  
087 \*/  
088 @Override  
089 public boolean add(final E object) {  
090 if(comparator() == null && !(object instanceof Comparable)) {  
091 if (object == null) {  
092 throw new NullPointerException();  
093 }  
094 throw new IllegalArgumentException("Objects of type " + object.getClass() + " cannot be added to " +  
095 "a naturally ordered TreeBag as it does not implement Comparable");  
096 }  
097 return super.add(object);  
098 }  
099  
100 //-----------------------------------------------------------------------  
101  
102 @Override  
103 public E first() {  
104 return getMap().firstKey();  
105 }  
106  
107 @Override  
108 public E last() {  
109 return getMap().lastKey();  
110 }  
111  
112 @Override  
113 public Comparator<? super E> comparator() {  
114 return getMap().comparator();  
115 }  
116  
117 @Override  
118 protected SortedMap<E, AbstractMapBag.MutableInteger> getMap() {  
119 return (SortedMap<E, AbstractMapBag.MutableInteger>) super.getMap();  
120 }  
121  
122 //-----------------------------------------------------------------------  
123 /\*\*  
124 \* Write the bag out using a custom routine.  
125 \*  
126 \* @param out the output stream  
127 \* @throws IOException if an error occurs while writing to the stream  
128 \*/  
129 private void writeObject(final ObjectOutputStream out) throws IOException {  
130 out.defaultWriteObject();  
131 out.writeObject(comparator());  
132 super.doWriteObject(out);  
133 }  
134  
135 /\*\*  
136 \* Read the bag in using a custom routine.  
137 \*  
138 \* @param in the input stream  
139 \* @throws IOException if an error occurs while reading from the stream  
140 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
141 \*/  
142 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
143 in.defaultReadObject();  
144 @SuppressWarnings("unchecked") // This will fail at runtime if the stream is incorrect  
145 final Comparator<? super E> comp = (Comparator<? super E>) in.readObject();  
146 super.doReadObject(new TreeMap<E, MutableInteger>(comp), in);  
147 }  
148  
149}