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.multiset;  
018  
019import java.util.Set;  
020  
021import org.apache.commons.collections4.MultiSet;  
022import org.apache.commons.collections4.collection.SynchronizedCollection;  
023  
024/\*\*  
025 \* Decorates another {@link MultiSet} to synchronize its behaviour  
026 \* for a multi-threaded environment.  
027 \* <p>  
028 \* Methods are synchronized, then forwarded to the decorated multiset.  
029 \* Iterators must be separately synchronized around the loop.  
030 \* </p>  
031 \*  
032 \* @param <E> the type held in the multiset  
033 \* @since 4.1  
034 \*/  
035public class SynchronizedMultiSet<E> extends SynchronizedCollection<E> implements MultiSet<E> {  
036  
037 /\*\* Serialization version \*/  
038 private static final long serialVersionUID = 20150629L;  
039  
040 /\*\*  
041 \* Factory method to create a synchronized multiset.  
042 \*  
043 \* @param <E> the type of the elements in the multiset  
044 \* @param multiset the multiset to decorate, must not be null  
045 \* @return a new synchronized MultiSet  
046 \* @throws NullPointerException if multiset is null  
047 \*/  
048 public static <E> SynchronizedMultiSet<E> synchronizedMultiSet(final MultiSet<E> multiset) {  
049 return new SynchronizedMultiSet<>(multiset);  
050 }  
051  
052 //-----------------------------------------------------------------------  
053 /\*\*  
054 \* Constructor that wraps (not copies).  
055 \*  
056 \* @param multiset the multiset to decorate, must not be null  
057 \* @throws NullPointerException if multiset is null  
058 \*/  
059 protected SynchronizedMultiSet(final MultiSet<E> multiset) {  
060 super(multiset);  
061 }  
062  
063 /\*\*  
064 \* Constructor that wraps (not copies).  
065 \*  
066 \* @param multiset the multiset to decorate, must not be null  
067 \* @param lock the lock to use, must not be null  
068 \* @throws NullPointerException if multiset or lock is null  
069 \*/  
070 protected SynchronizedMultiSet(final MultiSet<E> multiset, final Object lock) {  
071 super(multiset, lock);  
072 }  
073  
074 /\*\*  
075 \* Gets the multiset being decorated.  
076 \*  
077 \* @return the decorated multiset  
078 \*/  
079 @Override  
080 protected MultiSet<E> decorated() {  
081 return (MultiSet<E>) super.decorated();  
082 }  
083  
084 @Override  
085 public boolean equals(final Object object) {  
086 if (object == this) {  
087 return true;  
088 }  
089 synchronized (lock) {  
090 return decorated().equals(object);  
091 }  
092 }  
093  
094 @Override  
095 public int hashCode() {  
096 synchronized (lock) {  
097 return decorated().hashCode();  
098 }  
099 }  
100  
101 //-----------------------------------------------------------------------  
102  
103 @Override  
104 public int add(final E object, final int count) {  
105 synchronized (lock) {  
106 return decorated().add(object, count);  
107 }  
108 }  
109  
110 @Override  
111 public int remove(final Object object, final int count) {  
112 synchronized (lock) {  
113 return decorated().remove(object, count);  
114 }  
115 }  
116  
117 @Override  
118 public int getCount(final Object object) {  
119 synchronized (lock) {  
120 return decorated().getCount(object);  
121 }  
122 }  
123  
124 @Override  
125 public int setCount(final E object, final int count) {  
126 synchronized (lock) {  
127 return decorated().setCount(object, count);  
128 }  
129 }  
130  
131 @Override  
132 public Set<E> uniqueSet() {  
133 synchronized (lock) {  
134 final Set<E> set = decorated().uniqueSet();  
135 return new SynchronizedSet<>(set, lock);  
136 }  
137 }  
138  
139 @Override  
140 public Set<Entry<E>> entrySet() {  
141 synchronized (lock) {  
142 final Set<MultiSet.Entry<E>> set = decorated().entrySet();  
143 return new SynchronizedSet<>(set, lock);  
144 }  
145 }  
146  
147 //-----------------------------------------------------------------------  
148 /\*\*  
149 \* Synchronized Set for the MultiSet class.  
150 \*/  
151 static class SynchronizedSet<T> extends SynchronizedCollection<T> implements Set<T> {  
152 /\*\* Serialization version \*/  
153 private static final long serialVersionUID = 20150629L;  
154  
155 /\*\*  
156 \* Constructor.  
157 \* @param set the set to decorate  
158 \* @param lock the lock to use, shared with the multiset  
159 \*/  
160 SynchronizedSet(final Set<T> set, final Object lock) {  
161 super(set, lock);  
162 }  
163 }  
164  
165}