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.AbstractCollectionDecorator;  
023  
024/\*\*  
025 \* Decorates another <code>MultSet</code> to provide additional behaviour.  
026 \* <p>  
027 \* Methods are forwarded directly to the decorated multiset.  
028 \* </p>  
029 \*  
030 \* @param <E> the type held in the multiset  
031 \* @since 4.1  
032 \*/  
033public abstract class AbstractMultiSetDecorator<E>  
034 extends AbstractCollectionDecorator<E> implements MultiSet<E> {  
035  
036 /\*\* Serialization version \*/  
037 private static final long serialVersionUID = 20150610L;  
038  
039 /\*\*  
040 \* Constructor only used in deserialization, do not use otherwise.  
041 \*/  
042 protected AbstractMultiSetDecorator() {  
043 super();  
044 }  
045  
046 /\*\*  
047 \* Constructor that wraps (not copies).  
048 \*  
049 \* @param multiset the multiset to decorate, must not be null  
050 \* @throws NullPointerException if multiset is null  
051 \*/  
052 protected AbstractMultiSetDecorator(final MultiSet<E> multiset) {  
053 super(multiset);  
054 }  
055  
056 /\*\*  
057 \* Gets the multiset being decorated.  
058 \*  
059 \* @return the decorated multiset  
060 \*/  
061 @Override  
062 protected MultiSet<E> decorated() {  
063 return (MultiSet<E>) super.decorated();  
064 }  
065  
066 @Override  
067 public boolean equals(final Object object) {  
068 return object == this || decorated().equals(object);  
069 }  
070  
071 @Override  
072 public int hashCode() {  
073 return decorated().hashCode();  
074 }  
075  
076 //-----------------------------------------------------------------------  
077  
078 @Override  
079 public int getCount(final Object object) {  
080 return decorated().getCount(object);  
081 }  
082  
083 @Override  
084 public int setCount(final E object, final int count) {  
085 return decorated().setCount(object, count);  
086 }  
087  
088 @Override  
089 public int add(final E object, final int count) {  
090 return decorated().add(object, count);  
091 }  
092  
093 @Override  
094 public int remove(final Object object, final int count) {  
095 return decorated().remove(object, count);  
096 }  
097  
098 @Override  
099 public Set<E> uniqueSet() {  
100 return decorated().uniqueSet();  
101 }  
102  
103 @Override  
104 public Set<Entry<E>> entrySet() {  
105 return decorated().entrySet();  
106 }  
107  
108}