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.multimap;  
018  
019import java.util.Collections;  
020import java.util.Map;  
021import java.util.Set;  
022  
023import org.apache.commons.collections4.SetUtils;  
024import org.apache.commons.collections4.SetValuedMap;  
025  
026/\*\*  
027 \* Abstract implementation of the {@link SetValuedMap} interface to simplify the  
028 \* creation of subclass implementations.  
029 \* <p>  
030 \* Subclasses specify a Map implementation to use as the internal storage and  
031 \* the Set implementation to use as values.  
032 \* </p>  
033 \*  
034 \* @param <K> the type of the keys in this map  
035 \* @param <V> the type of the values in this map  
036 \* @since 4.1  
037 \*/  
038public abstract class AbstractSetValuedMap<K, V> extends AbstractMultiValuedMap<K, V>  
039 implements SetValuedMap<K, V> {  
040  
041 /\*\*  
042 \* Constructor needed for subclass serialisation.  
043 \*/  
044 protected AbstractSetValuedMap() {  
045 super();  
046 }  
047  
048 /\*\*  
049 \* A constructor that wraps, not copies  
050 \*  
051 \* @param map the map to wrap, must not be null  
052 \* @throws NullPointerException if the map is null  
053 \*/  
054 protected AbstractSetValuedMap(final Map<K, ? extends Set<V>> map) {  
055 super(map);  
056 }  
057  
058 // -----------------------------------------------------------------------  
059 @Override  
060 @SuppressWarnings("unchecked")  
061 protected Map<K, Set<V>> getMap() {  
062 return (Map<K, Set<V>>) super.getMap();  
063 }  
064  
065 /\*\*  
066 \* Creates a new value collection using the provided factory.  
067 \* @return a new list  
068 \*/  
069 @Override  
070 protected abstract Set<V> createCollection();  
071  
072 // -----------------------------------------------------------------------  
073 /\*\*  
074 \* Gets the set of values associated with the specified key. This would  
075 \* return an empty set in case the mapping is not present  
076 \*  
077 \* @param key the key to retrieve  
078 \* @return the <code>Set</code> of values, will return an empty  
079 \* <code>Set</code> for no mapping  
080 \*/  
081 @Override  
082 public Set<V> get(final K key) {  
083 return wrappedCollection(key);  
084 }  
085  
086 @Override  
087 Set<V> wrappedCollection(final K key) {  
088 return new WrappedSet(key);  
089 }  
090  
091 /\*\*  
092 \* Removes all values associated with the specified key.  
093 \* <p>  
094 \* A subsequent <code>get(Object)</code> would return an empty set.  
095 \*  
096 \* @param key the key to remove values from  
097 \* @return the <code>Set</code> of values removed, will return an empty,  
098 \* unmodifiable set for no mapping found.  
099 \*/  
100 @Override  
101 public Set<V> remove(final Object key) {  
102 return SetUtils.emptyIfNull(getMap().remove(key));  
103 }  
104  
105 // -----------------------------------------------------------------------  
106 /\*\*  
107 \* Wrapped set to handle add and remove on the collection returned by  
108 \* {@code get(Object)}.  
109 \*/  
110 private class WrappedSet extends WrappedCollection implements Set<V> {  
111  
112 public WrappedSet(final K key) {  
113 super(key);  
114 }  
115  
116 @Override  
117 public boolean equals(final Object other) {  
118 final Set<V> set = (Set<V>) getMapping();  
119 if (set == null) {  
120 return Collections.emptySet().equals(other);  
121 }  
122 if (!(other instanceof Set)) {  
123 return false;  
124 }  
125 final Set<?> otherSet = (Set<?>) other;  
126 return SetUtils.isEqualSet(set, otherSet);  
127 }  
128  
129 @Override  
130 public int hashCode() {  
131 final Set<V> set = (Set<V>) getMapping();  
132 return SetUtils.hashCodeForSet(set);  
133 }  
134  
135 }  
136  
137}