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.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.io.Serializable;  
023import java.util.ArrayList;  
024import java.util.Collection;  
025import java.util.HashMap;  
026import java.util.Map;  
027  
028import org.apache.commons.collections4.MultiValuedMap;  
029  
030/\*\*  
031 \* Implements a {@code ListValuedMap}, using a {@link HashMap} to provide data  
032 \* storage and {@link ArrayList}s as value collections. This is the standard  
033 \* implementation of a ListValuedMap.  
034 \* <p>  
035 \* <strong>Note that ArrayListValuedHashMap is not synchronized and is not  
036 \* thread-safe.</strong> If you wish to use this map from multiple threads  
037 \* concurrently, you must use appropriate synchronization. This class may throw  
038 \* exceptions when accessed by concurrent threads without synchronization.  
039 \* </p>  
040 \*  
041 \* @param <K> the type of the keys in this map  
042 \* @param <V> the type of the values in this map  
043 \* @since 4.1  
044 \*/  
045public class ArrayListValuedHashMap<K, V> extends AbstractListValuedMap<K, V>  
046 implements Serializable {  
047  
048 /\*\* Serialization Version \*/  
049 private static final long serialVersionUID = 20151118L;  
050  
051 /\*\*  
052 \* The initial map capacity used when none specified in constructor.  
053 \*/  
054 private static final int DEFAULT\_INITIAL\_MAP\_CAPACITY = 16;  
055  
056 /\*\*  
057 \* The initial list capacity when using none specified in constructor.  
058 \*/  
059 private static final int DEFAULT\_INITIAL\_LIST\_CAPACITY = 3;  
060  
061 /\*\*  
062 \* The initial list capacity when creating a new value collection.  
063 \*/  
064 private final int initialListCapacity;  
065  
066 /\*\*  
067 \* Creates an empty ArrayListValuedHashMap with the default initial  
068 \* map capacity (16) and the default initial list capacity (3).  
069 \*/  
070 public ArrayListValuedHashMap() {  
071 this(DEFAULT\_INITIAL\_MAP\_CAPACITY, DEFAULT\_INITIAL\_LIST\_CAPACITY);  
072 }  
073  
074 /\*\*  
075 \* Creates an empty ArrayListValuedHashMap with the default initial  
076 \* map capacity (16) and the specified initial list capacity.  
077 \*  
078 \* @param initialListCapacity the initial capacity used for value collections  
079 \*/  
080 public ArrayListValuedHashMap(final int initialListCapacity) {  
081 this(DEFAULT\_INITIAL\_MAP\_CAPACITY, initialListCapacity);  
082 }  
083  
084 /\*\*  
085 \* Creates an empty ArrayListValuedHashMap with the specified initial  
086 \* map and list capacities.  
087 \*  
088 \* @param initialMapCapacity the initial hashmap capacity  
089 \* @param initialListCapacity the initial capacity used for value collections  
090 \*/  
091 public ArrayListValuedHashMap(final int initialMapCapacity, final int initialListCapacity) {  
092 super(new HashMap<K, ArrayList<V>>(initialMapCapacity));  
093 this.initialListCapacity = initialListCapacity;  
094 }  
095  
096 /\*\*  
097 \* Creates an ArrayListValuedHashMap copying all the mappings of the given map.  
098 \*  
099 \* @param map a <code>MultiValuedMap</code> to copy into this map  
100 \*/  
101 public ArrayListValuedHashMap(final MultiValuedMap<? extends K, ? extends V> map) {  
102 this(map.size(), DEFAULT\_INITIAL\_LIST\_CAPACITY);  
103 super.putAll(map);  
104 }  
105  
106 /\*\*  
107 \* Creates an ArrayListValuedHashMap copying all the mappings of the given map.  
108 \*  
109 \* @param map a <code>Map</code> to copy into this map  
110 \*/  
111 public ArrayListValuedHashMap(final Map<? extends K, ? extends V> map) {  
112 this(map.size(), DEFAULT\_INITIAL\_LIST\_CAPACITY);  
113 super.putAll(map);  
114 }  
115  
116 // -----------------------------------------------------------------------  
117 @Override  
118 protected ArrayList<V> createCollection() {  
119 return new ArrayList<>(initialListCapacity);  
120 }  
121  
122 // -----------------------------------------------------------------------  
123 /\*\*  
124 \* Trims the capacity of all value collections to their current size.  
125 \*/  
126 public void trimToSize() {  
127 for (final Collection<V> coll : getMap().values()) {  
128 final ArrayList<V> list = (ArrayList<V>) coll;  
129 list.trimToSize();  
130 }  
131 }  
132  
133 // -----------------------------------------------------------------------  
134 private void writeObject(final ObjectOutputStream oos) throws IOException {  
135 oos.defaultWriteObject();  
136 doWriteObject(oos);  
137 }  
138  
139 private void readObject(final ObjectInputStream ois) throws IOException, ClassNotFoundException {  
140 ois.defaultReadObject();  
141 setMap(new HashMap<K, ArrayList<V>>());  
142 doReadObject(ois);  
143 }  
144  
145}