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.bidimap;  
018  
019import java.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.io.Serializable;  
023import java.util.HashMap;  
024import java.util.Map;  
025  
026import org.apache.commons.collections4.BidiMap;  
027  
028/\*\*  
029 \* Implementation of {@link BidiMap} that uses two {@link HashMap} instances.  
030 \* <p>  
031 \* Two {@link HashMap} instances are used in this class.  
032 \* This provides fast lookups at the expense of storing two sets of map entries.  
033 \* Commons Collections would welcome the addition of a direct hash-based  
034 \* implementation of the {@link BidiMap} interface.  
035 \* </p>  
036 \* <p>  
037 \* NOTE: From Commons Collections 3.1, all subclasses will use {@link HashMap}  
038 \* and the flawed <code>createMap</code> method is ignored.  
039 \* </p>  
040 \*  
041 \* @param <K> the type of the keys in the map  
042 \* @param <V> the type of the values in the map  
043 \*  
044 \* @since 3.0  
045 \*/  
046public class DualHashBidiMap<K, V> extends AbstractDualBidiMap<K, V> implements Serializable {  
047  
048 /\*\* Ensure serialization compatibility \*/  
049 private static final long serialVersionUID = 721969328361808L;  
050  
051 /\*\*  
052 \* Creates an empty <code>HashBidiMap</code>.  
053 \*/  
054 public DualHashBidiMap() {  
055 super(new HashMap<K, V>(), new HashMap<V, K>());  
056 }  
057  
058 /\*\*  
059 \* Constructs a <code>HashBidiMap</code> and copies the mappings from  
060 \* specified <code>Map</code>.  
061 \*  
062 \* @param map the map whose mappings are to be placed in this map  
063 \*/  
064 public DualHashBidiMap(final Map<? extends K, ? extends V> map) {  
065 super(new HashMap<K, V>(), new HashMap<V, K>());  
066 putAll(map);  
067 }  
068  
069 /\*\*  
070 \* Constructs a <code>HashBidiMap</code> that decorates the specified maps.  
071 \*  
072 \* @param normalMap the normal direction map  
073 \* @param reverseMap the reverse direction map  
074 \* @param inverseBidiMap the inverse BidiMap  
075 \*/  
076 protected DualHashBidiMap(final Map<K, V> normalMap, final Map<V, K> reverseMap,  
077 final BidiMap<V, K> inverseBidiMap) {  
078 super(normalMap, reverseMap, inverseBidiMap);  
079 }  
080  
081 /\*\*  
082 \* Creates a new instance of this object.  
083 \*  
084 \* @param normalMap the normal direction map  
085 \* @param reverseMap the reverse direction map  
086 \* @param inverseBidiMap the inverse BidiMap  
087 \* @return new bidi map  
088 \*/  
089 @Override  
090 protected BidiMap<V, K> createBidiMap(final Map<V, K> normalMap, final Map<K, V> reverseMap,  
091 final BidiMap<K, V> inverseBidiMap) {  
092 return new DualHashBidiMap<>(normalMap, reverseMap, inverseBidiMap);  
093 }  
094  
095 // Serialization  
096 //-----------------------------------------------------------------------  
097 private void writeObject(final ObjectOutputStream out) throws IOException {  
098 out.defaultWriteObject();  
099 out.writeObject(normalMap);  
100 }  
101  
102 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
103 in.defaultReadObject();  
104 normalMap = new HashMap<>();  
105 reverseMap = new HashMap<>();  
106 @SuppressWarnings("unchecked") // will fail at runtime if stream is incorrect  
107 final Map<K, V> map = (Map<K, V>) in.readObject();  
108 putAll(map);  
109 }  
110  
111}