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.LinkedHashMap;  
024import java.util.Map;  
025  
026import org.apache.commons.collections4.BidiMap;  
027  
028/\*\*  
029 \* Implementation of <code>BidiMap</code> that uses two <code>LinkedHashMap</code> instances.  
030 \* <p>  
031 \* Two <code>LinkedHashMap</code> instances are used in this class.  
032 \* This provides fast lookups at the expense of storing two sets of map entries and two linked lists.  
033 \* </p>  
034 \*  
035 \* @param <K> the type of the keys in the map  
036 \* @param <V> the type of the values in the map  
037 \*  
038 \* @since 4.0  
039 \*/  
040public class DualLinkedHashBidiMap<K, V> extends AbstractDualBidiMap<K, V> implements Serializable {  
041  
042 /\*\* Ensure serialization compatibility \*/  
043 private static final long serialVersionUID = 721969328361810L;  
044  
045 /\*\*  
046 \* Creates an empty <code>HashBidiMap</code>.  
047 \*/  
048 public DualLinkedHashBidiMap() {  
049 super(new LinkedHashMap<K, V>(), new LinkedHashMap<V, K>());  
050 }  
051  
052 /\*\*  
053 \* Constructs a <code>LinkedHashBidiMap</code> and copies the mappings from  
054 \* specified <code>Map</code>.  
055 \*  
056 \* @param map the map whose mappings are to be placed in this map  
057 \*/  
058 public DualLinkedHashBidiMap(final Map<? extends K, ? extends V> map) {  
059 super(new LinkedHashMap<K, V>(), new LinkedHashMap<V, K>());  
060 putAll(map);  
061 }  
062  
063 /\*\*  
064 \* Constructs a <code>LinkedHashBidiMap</code> that decorates the specified maps.  
065 \*  
066 \* @param normalMap the normal direction map  
067 \* @param reverseMap the reverse direction map  
068 \* @param inverseBidiMap the inverse BidiMap  
069 \*/  
070 protected DualLinkedHashBidiMap(final Map<K, V> normalMap, final Map<V, K> reverseMap,  
071 final BidiMap<V, K> inverseBidiMap) {  
072 super(normalMap, reverseMap, inverseBidiMap);  
073 }  
074  
075 /\*\*  
076 \* Creates a new instance of this object.  
077 \*  
078 \* @param normalMap the normal direction map  
079 \* @param reverseMap the reverse direction map  
080 \* @param inverseBidiMap the inverse BidiMap  
081 \* @return new bidi map  
082 \*/  
083 @Override  
084 protected BidiMap<V, K> createBidiMap(final Map<V, K> normalMap, final Map<K, V> reverseMap,  
085 final BidiMap<K, V> inverseBidiMap) {  
086 return new DualLinkedHashBidiMap<>(normalMap, reverseMap, inverseBidiMap);  
087 }  
088  
089 // Serialization  
090 //-----------------------------------------------------------------------  
091 private void writeObject(final ObjectOutputStream out) throws IOException {  
092 out.defaultWriteObject();  
093 out.writeObject(normalMap);  
094 }  
095  
096 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
097 in.defaultReadObject();  
098 normalMap = new LinkedHashMap<>();  
099 reverseMap = new LinkedHashMap<>();  
100 @SuppressWarnings("unchecked") // will fail at runtime if stream is incorrect  
101 final Map<K, V> map = (Map<K, V>) in.readObject();  
102 putAll(map);  
103 }  
104}