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.util.Set;  
020  
021import org.apache.commons.collections4.BidiMap;  
022import org.apache.commons.collections4.MapIterator;  
023import org.apache.commons.collections4.map.AbstractMapDecorator;  
024  
025/\*\*  
026 \* Provides a base decorator that enables additional functionality to be added  
027 \* to a BidiMap via decoration.  
028 \* <p>  
029 \* Methods are forwarded directly to the decorated map.  
030 \* </p>  
031 \* <p>  
032 \* This implementation does not perform any special processing with the map views.  
033 \* Instead it simply returns the set/collection from the wrapped map. This may be  
034 \* undesirable, for example if you are trying to write a validating implementation  
035 \* it would provide a loophole around the validation.  
036 \* But, you might want that loophole, so this class is kept simple.  
037 \* </p>  
038 \*  
039 \* @param <K> the type of the keys in this map  
040 \* @param <V> the type of the values in this map  
041 \* @since 3.0  
042 \*/  
043public abstract class AbstractBidiMapDecorator<K, V>  
044 extends AbstractMapDecorator<K, V> implements BidiMap<K, V> {  
045  
046 /\*\*  
047 \* Constructor that wraps (not copies).  
048 \*  
049 \* @param map the map to decorate, must not be null  
050 \* @throws NullPointerException if the collection is null  
051 \*/  
052 protected AbstractBidiMapDecorator(final BidiMap<K, V> map) {  
053 super(map);  
054 }  
055  
056 /\*\*  
057 \* Gets the map being decorated.  
058 \*  
059 \* @return the decorated map  
060 \*/  
061 @Override  
062 protected BidiMap<K, V> decorated() {  
063 return (BidiMap<K, V>) super.decorated();  
064 }  
065  
066 //-----------------------------------------------------------------------  
067 @Override  
068 public MapIterator<K, V> mapIterator() {  
069 return decorated().mapIterator();  
070 }  
071  
072 @Override  
073 public K getKey(final Object value) {  
074 return decorated().getKey(value);  
075 }  
076  
077 @Override  
078 public K removeValue(final Object value) {  
079 return decorated().removeValue(value);  
080 }  
081  
082 @Override  
083 public BidiMap<V, K> inverseBidiMap() {  
084 return decorated().inverseBidiMap();  
085 }  
086  
087 @Override  
088 public Set<V> values() {  
089 return decorated().values();  
090 }  
091  
092}