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.map;  
018  
019import org.apache.commons.collections4.OrderedMap;  
020import org.apache.commons.collections4.OrderedMapIterator;  
021  
022/\*\*  
023 \* Provides a base decorator that enables additional functionality to be added  
024 \* to an OrderedMap via decoration.  
025 \* <p>  
026 \* Methods are forwarded directly to the decorated map.  
027 \* </p>  
028 \* <p>  
029 \* This implementation does not perform any special processing with the map views.  
030 \* Instead it simply returns the set/collection from the wrapped map. This may be  
031 \* undesirable, for example if you are trying to write a validating implementation  
032 \* it would provide a loophole around the validation.  
033 \* But, you might want that loophole, so this class is kept simple.  
034 \* </p>  
035 \*  
036 \* @param <K> the type of the keys in this map  
037 \* @param <V> the type of the values in this map  
038 \* @since 3.0  
039 \*/  
040public abstract class AbstractOrderedMapDecorator<K, V> extends AbstractMapDecorator<K, V>  
041 implements OrderedMap<K, V> {  
042  
043 /\*\*  
044 \* Constructor only used in deserialization, do not use otherwise.  
045 \* @since 3.1  
046 \*/  
047 protected AbstractOrderedMapDecorator() {  
048 super();  
049 }  
050  
051 /\*\*  
052 \* Constructor that wraps (not copies).  
053 \*  
054 \* @param map the map to decorate, must not be null  
055 \* @throws NullPointerException if the map is null  
056 \*/  
057 public AbstractOrderedMapDecorator(final OrderedMap<K, V> map) {  
058 super(map);  
059 }  
060  
061 /\*\*  
062 \* Gets the map being decorated.  
063 \*  
064 \* @return the decorated map  
065 \*/  
066 @Override  
067 protected OrderedMap<K, V> decorated() {  
068 return (OrderedMap<K, V>) super.decorated();  
069 }  
070  
071 //-----------------------------------------------------------------------  
072 @Override  
073 public K firstKey() {  
074 return decorated().firstKey();  
075 }  
076  
077 @Override  
078 public K lastKey() {  
079 return decorated().lastKey();  
080 }  
081  
082 @Override  
083 public K nextKey(final K key) {  
084 return decorated().nextKey(key);  
085 }  
086  
087 @Override  
088 public K previousKey(final K key) {  
089 return decorated().previousKey(key);  
090 }  
091  
092 @Override  
093 public OrderedMapIterator<K, V> mapIterator() {  
094 return decorated().mapIterator();  
095 }  
096  
097}