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.splitmap;  
018  
019import java.util.Collection;  
020import java.util.Map;  
021import java.util.Set;  
022  
023import org.apache.commons.collections4.IterableGet;  
024import org.apache.commons.collections4.MapIterator;  
025import org.apache.commons.collections4.map.EntrySetToMapIteratorAdapter;  
026  
027/\*\*  
028 \* {@link IterableGet} that uses a {@link Map}<K, V> for the  
029 \* {@link org.apache.commons.collections4.Get Get}<K, V> implementation.  
030 \*  
031 \* @param <K> the type of the keys in this map  
032 \* @param <V> the type of the values in this map  
033 \* @since 4.0  
034 \*/  
035public class AbstractIterableGetMapDecorator<K, V> implements IterableGet<K, V> {  
036  
037 /\*\* The map to decorate \*/  
038 transient Map<K, V> map;  
039  
040 /\*\*  
041 \* Create a new AbstractSplitMapDecorator.  
042 \* @param map the map to decorate, must not be null  
043 \* @throws NullPointerException if map is null  
044 \*/  
045 public AbstractIterableGetMapDecorator(final Map<K, V> map) {  
046 if (map == null) {  
047 throw new NullPointerException("Map must not be null.");  
048 }  
049 this.map = map;  
050 }  
051  
052 /\*\*  
053 \* Constructor only used in deserialization, do not use otherwise.  
054 \*/  
055 protected AbstractIterableGetMapDecorator() {  
056 super();  
057 }  
058  
059 /\*\*  
060 \* Gets the map being decorated.  
061 \*  
062 \* @return the decorated map  
063 \*/  
064 protected Map<K, V> decorated() {  
065 return map;  
066 }  
067  
068 @Override  
069 public boolean containsKey(final Object key) {  
070 return decorated().containsKey(key);  
071 }  
072  
073 @Override  
074 public boolean containsValue(final Object value) {  
075 return decorated().containsValue(value);  
076 }  
077  
078 @Override  
079 public Set<Map.Entry<K, V>> entrySet() {  
080 return decorated().entrySet();  
081 }  
082  
083 @Override  
084 public V get(final Object key) {  
085 return decorated().get(key);  
086 }  
087  
088 @Override  
089 public V remove(final Object key) {  
090 return decorated().remove(key);  
091 }  
092  
093 @Override  
094 public boolean isEmpty() {  
095 return decorated().isEmpty();  
096 }  
097  
098 @Override  
099 public Set<K> keySet() {  
100 return decorated().keySet();  
101 }  
102  
103 @Override  
104 public int size() {  
105 return decorated().size();  
106 }  
107  
108 @Override  
109 public Collection<V> values() {  
110 return decorated().values();  
111 }  
112  
113 /\*\*  
114 \* Get a MapIterator over this Get.  
115 \* @return MapIterator<K, V>  
116 \*/  
117 @Override  
118 public MapIterator<K, V> mapIterator() {  
119 return new EntrySetToMapIteratorAdapter<>(entrySet());  
120 }  
121  
122 @Override  
123 public boolean equals(final Object object) {  
124 if (object == this) {  
125 return true;  
126 }  
127 return decorated().equals(object);  
128 }  
129  
130 @Override  
131 public int hashCode() {  
132 return decorated().hashCode();  
133 }  
134  
135 @Override  
136 public String toString() {  
137 return decorated().toString();  
138 }  
139  
140}