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.keyvalue;  
018  
019import java.util.Map;  
020  
021import org.apache.commons.collections4.KeyValue;  
022  
023/\*\*  
024 \* A mutable <code>KeyValue</code> pair that does not implement  
025 \* {@link java.util.Map.Entry Map.Entry}.  
026 \* <p>  
027 \* Note that a <code>DefaultKeyValue</code> instance may not contain  
028 \* itself as a key or value.  
029 \* </p>  
030 \*  
031 \* @param <K> the type of keys  
032 \* @param <V> the type of values  
033 \* @since 3.0  
034 \*/  
035public class DefaultKeyValue<K, V> extends AbstractKeyValue<K, V> {  
036  
037 /\*\*  
038 \* Constructs a new pair with a null key and null value.  
039 \*/  
040 public DefaultKeyValue() {  
041 super(null, null);  
042 }  
043  
044 /\*\*  
045 \* Constructs a new pair with the specified key and given value.  
046 \*  
047 \* @param key the key for the entry, may be null  
048 \* @param value the value for the entry, may be null  
049 \*/  
050 public DefaultKeyValue(final K key, final V value) {  
051 super(key, value);  
052 }  
053  
054 /\*\*  
055 \* Constructs a new pair from the specified <code>KeyValue</code>.  
056 \*  
057 \* @param pair the pair to copy, must not be null  
058 \* @throws NullPointerException if the entry is null  
059 \*/  
060 public DefaultKeyValue(final KeyValue<? extends K, ? extends V> pair) {  
061 super(pair.getKey(), pair.getValue());  
062 }  
063  
064 /\*\*  
065 \* Constructs a new pair from the specified <code>Map.Entry</code>.  
066 \*  
067 \* @param entry the entry to copy, must not be null  
068 \* @throws NullPointerException if the entry is null  
069 \*/  
070 public DefaultKeyValue(final Map.Entry<? extends K, ? extends V> entry) {  
071 super(entry.getKey(), entry.getValue());  
072 }  
073  
074 //-----------------------------------------------------------------------  
075 /\*\*  
076 \* Sets the key.  
077 \*  
078 \* @param key the new key  
079 \* @return the old key  
080 \* @throws IllegalArgumentException if key is this object  
081 \*/  
082 @Override  
083 public K setKey(final K key) {  
084 if (key == this) {  
085 throw new IllegalArgumentException("DefaultKeyValue may not contain itself as a key.");  
086 }  
087  
088 return super.setKey(key);  
089 }  
090  
091 /\*\*  
092 \* Sets the value.  
093 \*  
094 \* @return the old value of the value  
095 \* @param value the new value  
096 \* @throws IllegalArgumentException if value is this object  
097 \*/  
098 @Override  
099 public V setValue(final V value) {  
100 if (value == this) {  
101 throw new IllegalArgumentException("DefaultKeyValue may not contain itself as a value.");  
102 }  
103  
104 return super.setValue(value);  
105 }  
106  
107 //-----------------------------------------------------------------------  
108 /\*\*  
109 \* Returns a new <code>Map.Entry</code> object with key and value from this pair.  
110 \*  
111 \* @return a MapEntry instance  
112 \*/  
113 public Map.Entry<K, V> toMapEntry() {  
114 return new DefaultMapEntry<>(this);  
115 }  
116  
117 //-----------------------------------------------------------------------  
118 /\*\*  
119 \* Compares this <code>Map.Entry</code> with another <code>Map.Entry</code>.  
120 \* <p>  
121 \* Returns true if the compared object is also a <code>DefaultKeyValue</code>,  
122 \* and its key and value are equal to this object's key and value.  
123 \*  
124 \* @param obj the object to compare to  
125 \* @return true if equal key and value  
126 \*/  
127 @Override  
128 public boolean equals(final Object obj) {  
129 if (obj == this) {  
130 return true;  
131 }  
132 if (obj instanceof DefaultKeyValue == false) {  
133 return false;  
134 }  
135  
136 final DefaultKeyValue<?, ?> other = (DefaultKeyValue<?, ?>) obj;  
137 return  
138 (getKey() == null ? other.getKey() == null : getKey().equals(other.getKey())) &&  
139 (getValue() == null ? other.getValue() == null : getValue().equals(other.getValue()));  
140 }  
141  
142 /\*\*  
143 \* Gets a hashCode compatible with the equals method.  
144 \* <p>  
145 \* Implemented per API documentation of {@link java.util.Map.Entry#hashCode()},  
146 \* however subclasses may override this.  
147 \*  
148 \* @return a suitable hash code  
149 \*/  
150 @Override  
151 public int hashCode() {  
152 return (getKey() == null ? 0 : getKey().hashCode()) ^  
153 (getValue() == null ? 0 : getValue().hashCode());  
154 }  
155  
156}