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 java.io.IOException;  
020import java.io.ObjectInputStream;  
021import java.io.ObjectOutputStream;  
022import java.io.Serializable;  
023import java.util.Iterator;  
024import java.util.Map;  
025  
026import org.apache.commons.collections4.Predicate;  
027  
028/\*\*  
029 \* Decorates another <code>Map</code> to validate that additions  
030 \* match a specified predicate.  
031 \* <p>  
032 \* This map exists to provide validation for the decorated map.  
033 \* It is normally created to decorate an empty map.  
034 \* If an object cannot be added to the map, an IllegalArgumentException is thrown.  
035 \* </p>  
036 \* <p>  
037 \* One usage would be to ensure that no null keys are added to the map.  
038 \* </p>  
039 \* <pre>Map map = PredicatedSet.decorate(new HashMap(), NotNullPredicate.INSTANCE, null);</pre>  
040 \* <p>  
041 \* <strong>Note that PredicatedMap is not synchronized and is not thread-safe.</strong>  
042 \* If you wish to use this map from multiple threads concurrently, you must use  
043 \* appropriate synchronization. The simplest approach is to wrap this map  
044 \* using {@link java.util.Collections#synchronizedMap(Map)}. This class may throw  
045 \* exceptions when accessed by concurrent threads without synchronization.  
046 \* </p>  
047 \* <p>  
048 \* This class is Serializable from Commons Collections 3.1.  
049 \* </p>  
050 \*  
051 \* @param <K> the type of the keys in this map  
052 \* @param <V> the type of the values in this map  
053 \* @since 3.0  
054 \*/  
055public class PredicatedMap<K, V>  
056 extends AbstractInputCheckedMapDecorator<K, V>  
057 implements Serializable {  
058  
059 /\*\* Serialization version \*/  
060 private static final long serialVersionUID = 7412622456128415156L;  
061  
062 /\*\* The key predicate to use \*/  
063 protected final Predicate<? super K> keyPredicate;  
064  
065 /\*\* The value predicate to use \*/  
066 protected final Predicate<? super V> valuePredicate;  
067  
068 /\*\*  
069 \* Factory method to create a predicated (validating) map.  
070 \* <p>  
071 \* If there are any elements already in the list being decorated, they  
072 \* are validated.  
073 \*  
074 \* @param <K> the key type  
075 \* @param <V> the value type  
076 \* @param map the map to decorate, must not be null  
077 \* @param keyPredicate the predicate to validate the keys, null means no check  
078 \* @param valuePredicate the predicate to validate to values, null means no check  
079 \* @return a new predicated map  
080 \* @throws NullPointerException if the map is null  
081 \* @since 4.0  
082 \*/  
083 public static <K, V> PredicatedMap<K, V> predicatedMap(final Map<K, V> map,  
084 final Predicate<? super K> keyPredicate,  
085 final Predicate<? super V> valuePredicate) {  
086 return new PredicatedMap<>(map, keyPredicate, valuePredicate);  
087 }  
088  
089 //-----------------------------------------------------------------------  
090 /\*\*  
091 \* Constructor that wraps (not copies).  
092 \*  
093 \* @param map the map to decorate, must not be null  
094 \* @param keyPredicate the predicate to validate the keys, null means no check  
095 \* @param valuePredicate the predicate to validate to values, null means no check  
096 \* @throws NullPointerException if the map is null  
097 \*/  
098 protected PredicatedMap(final Map<K, V> map, final Predicate<? super K> keyPredicate,  
099 final Predicate<? super V> valuePredicate) {  
100 super(map);  
101 this.keyPredicate = keyPredicate;  
102 this.valuePredicate = valuePredicate;  
103  
104 final Iterator<Map.Entry<K, V>> it = map.entrySet().iterator();  
105 while (it.hasNext()) {  
106 final Map.Entry<K, V> entry = it.next();  
107 validate(entry.getKey(), entry.getValue());  
108 }  
109 }  
110  
111 //-----------------------------------------------------------------------  
112 /\*\*  
113 \* Write the map out using a custom routine.  
114 \*  
115 \* @param out the output stream  
116 \* @throws IOException if an error occurs while writing to the stream  
117 \* @since 3.1  
118 \*/  
119 private void writeObject(final ObjectOutputStream out) throws IOException {  
120 out.defaultWriteObject();  
121 out.writeObject(map);  
122 }  
123  
124 /\*\*  
125 \* Read the map in using a custom routine.  
126 \*  
127 \* @param in the input stream  
128 \* @throws IOException if an error occurs while reading from the stream  
129 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
130 \* @since 3.1  
131 \*/  
132 @SuppressWarnings("unchecked") // (1) should only fail if input stream is incorrect  
133 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
134 in.defaultReadObject();  
135 map = (Map<K, V>) in.readObject(); // (1)  
136 }  
137  
138 //-----------------------------------------------------------------------  
139 /\*\*  
140 \* Validates a key value pair.  
141 \*  
142 \* @param key the key to validate  
143 \* @param value the value to validate  
144 \* @throws IllegalArgumentException if invalid  
145 \*/  
146 protected void validate(final K key, final V value) {  
147 if (keyPredicate != null && keyPredicate.evaluate(key) == false) {  
148 throw new IllegalArgumentException("Cannot add key - Predicate rejected it");  
149 }  
150 if (valuePredicate != null && valuePredicate.evaluate(value) == false) {  
151 throw new IllegalArgumentException("Cannot add value - Predicate rejected it");  
152 }  
153 }  
154  
155 /\*\*  
156 \* Override to validate an object set into the map via <code>setValue</code>.  
157 \*  
158 \* @param value the value to validate  
159 \* @return the value itself  
160 \* @throws IllegalArgumentException if invalid  
161 \* @since 3.1  
162 \*/  
163 @Override  
164 protected V checkSetValue(final V value) {  
165 if (valuePredicate.evaluate(value) == false) {  
166 throw new IllegalArgumentException("Cannot set value - Predicate rejected it");  
167 }  
168 return value;  
169 }  
170  
171 /\*\*  
172 \* Override to only return true when there is a value transformer.  
173 \*  
174 \* @return true if a value predicate is in use  
175 \* @since 3.1  
176 \*/  
177 @Override  
178 protected boolean isSetValueChecking() {  
179 return valuePredicate != null;  
180 }  
181  
182 //-----------------------------------------------------------------------  
183 @Override  
184 public V put(final K key, final V value) {  
185 validate(key, value);  
186 return map.put(key, value);  
187 }  
188  
189 @Override  
190 public void putAll(final Map<? extends K, ? extends V> mapToCopy) {  
191 for (final Map.Entry<? extends K, ? extends V> entry : mapToCopy.entrySet()) {  
192 validate(entry.getKey(), entry.getValue());  
193 }  
194 super.putAll(mapToCopy);  
195 }  
196  
197}