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.Map;  
024  
025import org.apache.commons.collections4.Factory;  
026import org.apache.commons.collections4.Transformer;  
027import org.apache.commons.collections4.functors.FactoryTransformer;  
028  
029/\*\*  
030 \* Decorates another <code>Map</code> to create objects in the map on demand.  
031 \* <p>  
032 \* When the {@link #get(Object)} method is called with a key that does not  
033 \* exist in the map, the factory is used to create the object. The created  
034 \* object will be added to the map using the requested key.  
035 \* </p>  
036 \* <p>  
037 \* For instance:  
038 \* </p>  
039 \* <pre>  
040 \* Factory<Date> factory = new Factory<Date>() {  
041 \* public Date create() {  
042 \* return new Date();  
043 \* }  
044 \* }  
045 \* Map<String, Date> lazy = LazyMap.lazyMap(new HashMap<String, Date>(), factory);  
046 \* Date date = lazy.get("NOW");  
047 \* </pre>  
048 \*  
049 \* <p>  
050 \* After the above code is executed, <code>date</code> will refer to  
051 \* a new <code>Date</code> instance. Furthermore, that <code>Date</code>  
052 \* instance is mapped to the "NOW" key in the map.  
053 \* </p>  
054 \* <p>  
055 \* <strong>Note that LazyMap is not synchronized and is not thread-safe.</strong>  
056 \* If you wish to use this map from multiple threads concurrently, you must use  
057 \* appropriate synchronization. The simplest approach is to wrap this map  
058 \* using {@link java.util.Collections#synchronizedMap(Map)}. This class may throw  
059 \* exceptions when accessed by concurrent threads without synchronization.  
060 \* </p>  
061 \* <p>  
062 \* This class is Serializable from Commons Collections 3.1.  
063 \* </p>  
064 \*  
065 \* @param <K> the type of the keys in this map  
066 \* @param <V> the type of the values in this map  
067 \* @since 3.0  
068 \*/  
069public class LazyMap<K, V> extends AbstractMapDecorator<K, V> implements Serializable {  
070  
071 /\*\* Serialization version \*/  
072 private static final long serialVersionUID = 7990956402564206740L;  
073  
074 /\*\* The factory to use to construct elements \*/  
075 protected final Transformer<? super K, ? extends V> factory;  
076  
077 /\*\*  
078 \* Factory method to create a lazily instantiated map.  
079 \*  
080 \* @param <K> the key type  
081 \* @param <V> the value type  
082 \* @param map the map to decorate, must not be null  
083 \* @param factory the factory to use, must not be null  
084 \* @return a new lazy map  
085 \* @throws NullPointerException if map or factory is null  
086 \* @since 4.0  
087 \*/  
088 public static <K, V> LazyMap<K, V> lazyMap(final Map<K, V> map, final Factory< ? extends V> factory) {  
089 return new LazyMap<>(map, factory);  
090 }  
091  
092 /\*\*  
093 \* Factory method to create a lazily instantiated map.  
094 \*  
095 \* @param <K> the key type  
096 \* @param <V> the value type  
097 \* @param map the map to decorate, must not be null  
098 \* @param factory the factory to use, must not be null  
099 \* @return a new lazy map  
100 \* @throws NullPointerException if map or factory is null  
101 \* @since 4.0  
102 \*/  
103 public static <V, K> LazyMap<K, V> lazyMap(final Map<K, V> map, final Transformer<? super K, ? extends V> factory) {  
104 return new LazyMap<>(map, factory);  
105 }  
106  
107 //-----------------------------------------------------------------------  
108 /\*\*  
109 \* Constructor that wraps (not copies).  
110 \*  
111 \* @param map the map to decorate, must not be null  
112 \* @param factory the factory to use, must not be null  
113 \* @throws NullPointerException if map or factory is null  
114 \*/  
115 protected LazyMap(final Map<K,V> map, final Factory<? extends V> factory) {  
116 super(map);  
117 if (factory == null) {  
118 throw new NullPointerException("Factory must not be null");  
119 }  
120 this.factory = FactoryTransformer.factoryTransformer(factory);  
121 }  
122  
123 /\*\*  
124 \* Constructor that wraps (not copies).  
125 \*  
126 \* @param map the map to decorate, must not be null  
127 \* @param factory the factory to use, must not be null  
128 \* @throws NullPointerException if map or factory is null  
129 \*/  
130 protected LazyMap(final Map<K,V> map, final Transformer<? super K, ? extends V> factory) {  
131 super(map);  
132 if (factory == null) {  
133 throw new NullPointerException("Factory must not be null");  
134 }  
135 this.factory = factory;  
136 }  
137  
138 //-----------------------------------------------------------------------  
139 /\*\*  
140 \* Write the map out using a custom routine.  
141 \*  
142 \* @param out the output stream  
143 \* @throws IOException if an error occurs while writing to the stream  
144 \* @since 3.1  
145 \*/  
146 private void writeObject(final ObjectOutputStream out) throws IOException {  
147 out.defaultWriteObject();  
148 out.writeObject(map);  
149 }  
150  
151 /\*\*  
152 \* Read the map in using a custom routine.  
153 \*  
154 \* @param in the input stream  
155 \* @throws IOException if an error occurs while reading from the stream  
156 \* @throws ClassNotFoundException if an object read from the stream can not be loaded  
157 \* @since 3.1  
158 \*/  
159 @SuppressWarnings("unchecked")  
160 private void readObject(final ObjectInputStream in) throws IOException, ClassNotFoundException {  
161 in.defaultReadObject();  
162 map = (Map<K, V>) in.readObject();  
163 }  
164  
165 //-----------------------------------------------------------------------  
166 @Override  
167 public V get(final Object key) {  
168 // create value for key if key is not currently in the map  
169 if (map.containsKey(key) == false) {  
170 @SuppressWarnings("unchecked")  
171 final K castKey = (K) key;  
172 final V value = factory.transform(castKey);  
173 map.put(castKey, value);  
174 return value;  
175 }  
176 return map.get(key);  
177 }  
178  
179 // no need to wrap keySet, entrySet or values as they are views of  
180 // existing map entries - you can't do a map-style get on them.  
181}