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.functors;  
018  
019import java.io.Serializable;  
020import java.util.Map;  
021  
022import org.apache.commons.collections4.Predicate;  
023import org.apache.commons.collections4.Transformer;  
024  
025/\*\*  
026 \* Transformer implementation calls the transformer whose predicate returns true,  
027 \* like a switch statement.  
028 \*  
029 \* @since 3.0  
030 \*/  
031public class SwitchTransformer<I, O> implements Transformer<I, O>, Serializable {  
032  
033 /\*\* Serial version UID \*/  
034 private static final long serialVersionUID = -6404460890903469332L;  
035  
036 /\*\* The tests to consider \*/  
037 private final Predicate<? super I>[] iPredicates;  
038 /\*\* The matching transformers to call \*/  
039 private final Transformer<? super I, ? extends O>[] iTransformers;  
040 /\*\* The default transformer to call if no tests match \*/  
041 private final Transformer<? super I, ? extends O> iDefault;  
042  
043 /\*\*  
044 \* Factory method that performs validation and copies the parameter arrays.  
045 \*  
046 \* @param <I> the input type  
047 \* @param <O> the output type  
048 \* @param predicates array of predicates, cloned, no nulls  
049 \* @param transformers matching array of transformers, cloned, no nulls  
050 \* @param defaultTransformer the transformer to use if no match, null means return null  
051 \* @return the <code>chained</code> transformer  
052 \* @throws NullPointerException if array is null  
053 \* @throws NullPointerException if any element in the array is null  
054 \*/  
055 @SuppressWarnings("unchecked")  
056 public static <I, O> Transformer<I, O> switchTransformer(final Predicate<? super I>[] predicates,  
057 final Transformer<? super I, ? extends O>[] transformers,  
058 final Transformer<? super I, ? extends O> defaultTransformer) {  
059 FunctorUtils.validate(predicates);  
060 FunctorUtils.validate(transformers);  
061 if (predicates.length != transformers.length) {  
062 throw new IllegalArgumentException("The predicate and transformer arrays must be the same size");  
063 }  
064 if (predicates.length == 0) {  
065 return (Transformer<I, O>) (defaultTransformer == null ? ConstantTransformer.<I, O>nullTransformer() :  
066 defaultTransformer);  
067 }  
068 return new SwitchTransformer<>(predicates, transformers, defaultTransformer);  
069 }  
070  
071 /\*\*  
072 \* Create a new Transformer that calls one of the transformers depending  
073 \* on the predicates.  
074 \* <p>  
075 \* The Map consists of Predicate keys and Transformer values. A transformer  
076 \* is called if its matching predicate returns true. Each predicate is evaluated  
077 \* until one returns true. If no predicates evaluate to true, the default  
078 \* transformer is called. The default transformer is set in the map with a  
079 \* null key. The ordering is that of the iterator() method on the entryset  
080 \* collection of the map.  
081 \*  
082 \* @param <I> the input type  
083 \* @param <O> the output type  
084 \* @param map a map of predicates to transformers  
085 \* @return the <code>switch</code> transformer  
086 \* @throws NullPointerException if the map is null  
087 \* @throws NullPointerException if any transformer in the map is null  
088 \* @throws ClassCastException if the map elements are of the wrong type  
089 \*/  
090 @SuppressWarnings("unchecked")  
091 public static <I, O> Transformer<I, O> switchTransformer(  
092 final Map<? extends Predicate<? super I>, ? extends Transformer<? super I, ? extends O>> map) {  
093  
094 if (map == null) {  
095 throw new NullPointerException("The predicate and transformer map must not be null");  
096 }  
097 if (map.size() == 0) {  
098 return ConstantTransformer.<I, O>nullTransformer();  
099 }  
100 // convert to array like this to guarantee iterator() ordering  
101 final Transformer<? super I, ? extends O> defaultTransformer = map.remove(null);  
102 final int size = map.size();  
103 if (size == 0) {  
104 return (Transformer<I, O>) (defaultTransformer == null ? ConstantTransformer.<I, O>nullTransformer() :  
105 defaultTransformer);  
106 }  
107 final Transformer<? super I, ? extends O>[] transformers = new Transformer[size];  
108 final Predicate<? super I>[] preds = new Predicate[size];  
109 int i = 0;  
110 for (final Map.Entry<? extends Predicate<? super I>,  
111 ? extends Transformer<? super I, ? extends O>> entry : map.entrySet()) {  
112 preds[i] = entry.getKey();  
113 transformers[i] = entry.getValue();  
114 i++;  
115 }  
116 return new SwitchTransformer<>(false, preds, transformers, defaultTransformer);  
117 }  
118  
119 /\*\*  
120 \* Hidden constructor for the use by the static factory methods.  
121 \*  
122 \* @param clone if {@code true} the input arguments will be cloned  
123 \* @param predicates array of predicates, no nulls  
124 \* @param transformers matching array of transformers, no nulls  
125 \* @param defaultTransformer the transformer to use if no match, null means return null  
126 \*/  
127 @SuppressWarnings("unchecked")  
128 private SwitchTransformer(final boolean clone, final Predicate<? super I>[] predicates,  
129 final Transformer<? super I, ? extends O>[] transformers,  
130 final Transformer<? super I, ? extends O> defaultTransformer) {  
131 super();  
132 iPredicates = clone ? FunctorUtils.copy(predicates) : predicates;  
133 iTransformers = clone ? FunctorUtils.copy(transformers) : transformers;  
134 iDefault = (Transformer<? super I, ? extends O>) (defaultTransformer == null ?  
135 ConstantTransformer.<I, O>nullTransformer() : defaultTransformer);  
136 }  
137  
138 /\*\*  
139 \* Constructor that performs no validation.  
140 \* Use <code>switchTransformer</code> if you want that.  
141 \*  
142 \* @param predicates array of predicates, cloned, no nulls  
143 \* @param transformers matching array of transformers, cloned, no nulls  
144 \* @param defaultTransformer the transformer to use if no match, null means return null  
145 \*/  
146 public SwitchTransformer(final Predicate<? super I>[] predicates,  
147 final Transformer<? super I, ? extends O>[] transformers,  
148 final Transformer<? super I, ? extends O> defaultTransformer) {  
149 this(true, predicates, transformers, defaultTransformer);  
150 }  
151  
152 /\*\*  
153 \* Transforms the input to result by calling the transformer whose matching  
154 \* predicate returns true.  
155 \*  
156 \* @param input the input object to transform  
157 \* @return the transformed result  
158 \*/  
159 @Override  
160 public O transform(final I input) {  
161 for (int i = 0; i < iPredicates.length; i++) {  
162 if (iPredicates[i].evaluate(input) == true) {  
163 return iTransformers[i].transform(input);  
164 }  
165 }  
166 return iDefault.transform(input);  
167 }  
168  
169 /\*\*  
170 \* Gets the predicates.  
171 \*  
172 \* @return a copy of the predicates  
173 \* @since 3.1  
174 \*/  
175 public Predicate<? super I>[] getPredicates() {  
176 return FunctorUtils.<I>copy(iPredicates);  
177 }  
178  
179 /\*\*  
180 \* Gets the transformers.  
181 \*  
182 \* @return a copy of the transformers  
183 \* @since 3.1  
184 \*/  
185 public Transformer<? super I, ? extends O>[] getTransformers() {  
186 return FunctorUtils.<I, O>copy(iTransformers);  
187 }  
188  
189 /\*\*  
190 \* Gets the default transformer.  
191 \*  
192 \* @return the default transformer  
193 \* @since 3.1  
194 \*/  
195 public Transformer<? super I, ? extends O> getDefaultTransformer() {  
196 return iDefault;  
197 }  
198  
199}