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;  
018  
019import java.util.Collection;  
020import java.util.Map;  
021  
022import org.apache.commons.collections4.functors.ChainedClosure;  
023import org.apache.commons.collections4.functors.EqualPredicate;  
024import org.apache.commons.collections4.functors.ExceptionClosure;  
025import org.apache.commons.collections4.functors.ForClosure;  
026import org.apache.commons.collections4.functors.IfClosure;  
027import org.apache.commons.collections4.functors.InvokerTransformer;  
028import org.apache.commons.collections4.functors.NOPClosure;  
029import org.apache.commons.collections4.functors.SwitchClosure;  
030import org.apache.commons.collections4.functors.TransformerClosure;  
031import org.apache.commons.collections4.functors.WhileClosure;  
032  
033/\*\*  
034 \* <code>ClosureUtils</code> provides reference implementations and utilities  
035 \* for the Closure functor interface. The supplied closures are:  
036 \* <ul>  
037 \* <li>Invoker - invokes a method on the input object  
038 \* <li>For - repeatedly calls a closure for a fixed number of times  
039 \* <li>While - repeatedly calls a closure while a predicate is true  
040 \* <li>Chained - chains two or more closures together  
041 \* <li>If - calls one closure or another based on a predicate  
042 \* <li>Switch - calls one closure based on one or more predicates  
043 \* <li>SwitchMap - calls one closure looked up from a Map  
044 \* <li>Transformer - wraps a Transformer as a Closure  
045 \* <li>NOP - does nothing  
046 \* <li>Exception - always throws an exception  
047 \* </ul>  
048 \* <p>  
049 \* Since v4.1 only closures which are considered to be safe are  
050 \* Serializable. Closures considered to be unsafe for serialization are:  
051 \* </p>  
052 \* <ul>  
053 \* <li>Invoker  
054 \* <li>For  
055 \* <li>While  
056 \* </ul>  
057 \*  
058 \* @since 3.0  
059 \*/  
060public class ClosureUtils {  
061  
062 /\*\*  
063 \* This class is not normally instantiated.  
064 \*/  
065 private ClosureUtils() {}  
066  
067 /\*\*  
068 \* Gets a Closure that always throws an exception.  
069 \* This could be useful during testing as a placeholder.  
070 \*  
071 \* @see org.apache.commons.collections4.functors.ExceptionClosure  
072 \*  
073 \* @param <E> the type that the closure acts on  
074 \* @return the closure  
075 \*/  
076 public static <E> Closure<E> exceptionClosure() {  
077 return ExceptionClosure.<E>exceptionClosure();  
078 }  
079  
080 /\*\*  
081 \* Gets a Closure that will do nothing.  
082 \* This could be useful during testing as a placeholder.  
083 \*  
084 \* @see org.apache.commons.collections4.functors.NOPClosure  
085 \*  
086 \* @param <E> the type that the closure acts on  
087 \* @return the closure  
088 \*/  
089 public static <E> Closure<E> nopClosure() {  
090 return NOPClosure.<E>nopClosure();  
091 }  
092  
093 /\*\*  
094 \* Creates a Closure that calls a Transformer each time it is called.  
095 \* The transformer will be called using the closure's input object.  
096 \* The transformer's result will be ignored.  
097 \*  
098 \* @see org.apache.commons.collections4.functors.TransformerClosure  
099 \*  
100 \* @param <E> the type that the closure acts on  
101 \* @param transformer the transformer to run each time in the closure, null means nop  
102 \* @return the closure  
103 \*/  
104 public static <E> Closure<E> asClosure(final Transformer<? super E, ?> transformer) {  
105 return TransformerClosure.transformerClosure(transformer);  
106 }  
107  
108 /\*\*  
109 \* Creates a Closure that will call the closure <code>count</code> times.  
110 \* <p>  
111 \* A null closure or zero count returns the <code>NOPClosure</code>.  
112 \*  
113 \* @see org.apache.commons.collections4.functors.ForClosure  
114 \*  
115 \* @param <E> the type that the closure acts on  
116 \* @param count the number of times to loop  
117 \* @param closure the closure to call repeatedly  
118 \* @return the <code>for</code> closure  
119 \*/  
120 public static <E> Closure<E> forClosure(final int count, final Closure<? super E> closure) {  
121 return ForClosure.forClosure(count, closure);  
122 }  
123  
124 /\*\*  
125 \* Creates a Closure that will call the closure repeatedly until the  
126 \* predicate returns false.  
127 \*  
128 \* @see org.apache.commons.collections4.functors.WhileClosure  
129 \*  
130 \* @param <E> the type that the closure acts on  
131 \* @param predicate the predicate to use as an end of loop test, not null  
132 \* @param closure the closure to call repeatedly, not null  
133 \* @return the <code>while</code> closure  
134 \* @throws NullPointerException if either argument is null  
135 \*/  
136 public static <E> Closure<E> whileClosure(final Predicate<? super E> predicate, final Closure<? super E> closure) {  
137 return WhileClosure.<E>whileClosure(predicate, closure, false);  
138 }  
139  
140 /\*\*  
141 \* Creates a Closure that will call the closure once and then repeatedly  
142 \* until the predicate returns false.  
143 \*  
144 \* @see org.apache.commons.collections4.functors.WhileClosure  
145 \*  
146 \* @param <E> the type that the closure acts on  
147 \* @param closure the closure to call repeatedly, not null  
148 \* @param predicate the predicate to use as an end of loop test, not null  
149 \* @return the <code>do-while</code> closure  
150 \* @throws NullPointerException if either argument is null  
151 \*/  
152 public static <E> Closure<E> doWhileClosure(final Closure<? super E> closure,  
153 final Predicate<? super E> predicate) {  
154 return WhileClosure.<E>whileClosure(predicate, closure, true);  
155 }  
156  
157 /\*\*  
158 \* Creates a Closure that will invoke a specific method on the closure's  
159 \* input object by reflection.  
160 \*  
161 \* @see org.apache.commons.collections4.functors.InvokerTransformer  
162 \* @see org.apache.commons.collections4.functors.TransformerClosure  
163 \*  
164 \* @param <E> the type that the closure acts on  
165 \* @param methodName the name of the method  
166 \* @return the <code>invoker</code> closure  
167 \* @throws NullPointerException if the method name is null  
168 \*/  
169 public static <E> Closure<E> invokerClosure(final String methodName) {  
170 // reuse transformer as it has caching - this is lazy really, should have inner class here  
171 return asClosure(InvokerTransformer.<E, Object>invokerTransformer(methodName));  
172 }  
173  
174 /\*\*  
175 \* Creates a Closure that will invoke a specific method on the closure's  
176 \* input object by reflection.  
177 \*  
178 \* @see org.apache.commons.collections4.functors.InvokerTransformer  
179 \* @see org.apache.commons.collections4.functors.TransformerClosure  
180 \*  
181 \* @param <E> the type that the closure acts on  
182 \* @param methodName the name of the method  
183 \* @param paramTypes the parameter types  
184 \* @param args the arguments  
185 \* @return the <code>invoker</code> closure  
186 \* @throws NullPointerException if the method name is null  
187 \* @throws IllegalArgumentException if the paramTypes and args don't match  
188 \*/  
189 public static <E> Closure<E> invokerClosure(final String methodName, final Class<?>[] paramTypes,  
190 final Object[] args) {  
191 // reuse transformer as it has caching - this is lazy really, should have inner class here  
192 return asClosure(InvokerTransformer.<E, Object>invokerTransformer(methodName, paramTypes, args));  
193 }  
194  
195 /\*\*  
196 \* Create a new Closure that calls each closure in turn, passing the  
197 \* result into the next closure.  
198 \*  
199 \* @see org.apache.commons.collections4.functors.ChainedClosure  
200 \*  
201 \* @param <E> the type that the closure acts on  
202 \* @param closures an array of closures to chain  
203 \* @return the <code>chained</code> closure  
204 \* @throws NullPointerException if the closures array is null  
205 \* @throws NullPointerException if any closure in the array is null  
206 \*/  
207 public static <E> Closure<E> chainedClosure(final Closure<? super E>... closures) {  
208 return ChainedClosure.chainedClosure(closures);  
209 }  
210  
211 /\*\*  
212 \* Create a new Closure that calls each closure in turn, passing the  
213 \* result into the next closure. The ordering is that of the iterator()  
214 \* method on the collection.  
215 \*  
216 \* @see org.apache.commons.collections4.functors.ChainedClosure  
217 \*  
218 \* @param <E> the type that the closure acts on  
219 \* @param closures a collection of closures to chain  
220 \* @return the <code>chained</code> closure  
221 \* @throws NullPointerException if the closures collection is null  
222 \* @throws NullPointerException if any closure in the collection is null  
223 \* @throws IllegalArgumentException if the closures collection is empty  
224 \*/  
225 public static <E> Closure<E> chainedClosure(final Collection<? extends Closure<? super E>> closures) {  
226 return ChainedClosure.chainedClosure(closures);  
227 }  
228  
229 /\*\*  
230 \* Create a new Closure that calls another closure based on the  
231 \* result of the specified predicate.  
232 \*  
233 \* @see org.apache.commons.collections4.functors.IfClosure  
234 \*  
235 \* @param <E> the type that the closure acts on  
236 \* @param predicate the validating predicate  
237 \* @param trueClosure the closure called if the predicate is true  
238 \* @return the <code>if</code> closure  
239 \* @throws NullPointerException if the predicate or closure is null  
240 \* @since 3.2  
241 \*/  
242 public static <E> Closure<E> ifClosure(final Predicate<? super E> predicate,  
243 final Closure<? super E> trueClosure) {  
244 return IfClosure.<E>ifClosure(predicate, trueClosure);  
245 }  
246  
247 /\*\*  
248 \* Create a new Closure that calls one of two closures depending  
249 \* on the specified predicate.  
250 \*  
251 \* @see org.apache.commons.collections4.functors.IfClosure  
252 \*  
253 \* @param <E> the type that the closure acts on  
254 \* @param predicate the predicate to switch on  
255 \* @param trueClosure the closure called if the predicate is true  
256 \* @param falseClosure the closure called if the predicate is false  
257 \* @return the <code>switch</code> closure  
258 \* @throws NullPointerException if the predicate or either closure is null  
259 \*/  
260 public static <E> Closure<E> ifClosure(final Predicate<? super E> predicate,  
261 final Closure<? super E> trueClosure,  
262 final Closure<? super E> falseClosure) {  
263 return IfClosure.<E>ifClosure(predicate, trueClosure, falseClosure);  
264 }  
265  
266 /\*\*  
267 \* Create a new Closure that calls one of the closures depending  
268 \* on the predicates.  
269 \* <p>  
270 \* The closure at array location 0 is called if the predicate at array  
271 \* location 0 returned true. Each predicate is evaluated  
272 \* until one returns true.  
273 \*  
274 \* @see org.apache.commons.collections4.functors.SwitchClosure  
275 \*  
276 \* @param <E> the type that the closure acts on  
277 \* @param predicates an array of predicates to check, not null  
278 \* @param closures an array of closures to call, not null  
279 \* @return the <code>switch</code> closure  
280 \* @throws NullPointerException if the either array is null  
281 \* @throws NullPointerException if any element in the arrays is null  
282 \* @throws IllegalArgumentException if the arrays have different sizes  
283 \*/  
284 public static <E> Closure<E> switchClosure(final Predicate<? super E>[] predicates,  
285 final Closure<? super E>[] closures) {  
286 return SwitchClosure.<E>switchClosure(predicates, closures, null);  
287 }  
288  
289 /\*\*  
290 \* Create a new Closure that calls one of the closures depending  
291 \* on the predicates.  
292 \* <p>  
293 \* The closure at array location 0 is called if the predicate at array  
294 \* location 0 returned true. Each predicate is evaluated  
295 \* until one returns true. If no predicates evaluate to true, the default  
296 \* closure is called.  
297 \*  
298 \* @see org.apache.commons.collections4.functors.SwitchClosure  
299 \*  
300 \* @param <E> the type that the closure acts on  
301 \* @param predicates an array of predicates to check, not null  
302 \* @param closures an array of closures to call, not null  
303 \* @param defaultClosure the default to call if no predicate matches  
304 \* @return the <code>switch</code> closure  
305 \* @throws NullPointerException if the either array is null  
306 \* @throws NullPointerException if any element in the arrays is null  
307 \* @throws IllegalArgumentException if the arrays are different sizes  
308 \*/  
309 public static <E> Closure<E> switchClosure(final Predicate<? super E>[] predicates,  
310 final Closure<? super E>[] closures,  
311 final Closure<? super E> defaultClosure) {  
312 return SwitchClosure.<E>switchClosure(predicates, closures, defaultClosure);  
313 }  
314  
315 /\*\*  
316 \* Create a new Closure that calls one of the closures depending  
317 \* on the predicates.  
318 \* <p>  
319 \* The Map consists of Predicate keys and Closure values. A closure  
320 \* is called if its matching predicate returns true. Each predicate is evaluated  
321 \* until one returns true. If no predicates evaluate to true, the default  
322 \* closure is called. The default closure is set in the map with a  
323 \* null key. The ordering is that of the iterator() method on the entryset  
324 \* collection of the map.  
325 \*  
326 \* @see org.apache.commons.collections4.functors.SwitchClosure  
327 \*  
328 \* @param <E> the type that the closure acts on  
329 \* @param predicatesAndClosures a map of predicates to closures  
330 \* @return the <code>switch</code> closure  
331 \* @throws NullPointerException if the map is null  
332 \* @throws NullPointerException if any closure in the map is null  
333 \* @throws IllegalArgumentException if the map is empty  
334 \* @throws ClassCastException if the map elements are of the wrong type  
335 \*/  
336 public static <E> Closure<E> switchClosure(final Map<Predicate<E>, Closure<E>> predicatesAndClosures) {  
337 return SwitchClosure.switchClosure(predicatesAndClosures);  
338 }  
339  
340 /\*\*  
341 \* Create a new Closure that uses the input object as a key to find the  
342 \* closure to call.  
343 \* <p>  
344 \* The Map consists of object keys and Closure values. A closure  
345 \* is called if the input object equals the key. If there is no match, the  
346 \* default closure is called. The default closure is set in the map  
347 \* using a null key.  
348 \*  
349 \* @see org.apache.commons.collections4.functors.SwitchClosure  
350 \*  
351 \* @param <E> the type that the closure acts on  
352 \* @param objectsAndClosures a map of objects to closures  
353 \* @return the closure  
354 \* @throws NullPointerException if the map is null  
355 \* @throws NullPointerException if any closure in the map is null  
356 \* @throws IllegalArgumentException if the map is empty  
357 \*/  
358 @SuppressWarnings("unchecked")  
359 public static <E> Closure<E> switchMapClosure(final Map<? extends E, Closure<E>> objectsAndClosures) {  
360 if (objectsAndClosures == null) {  
361 throw new NullPointerException("The object and closure map must not be null");  
362 }  
363 final Closure<? super E> def = objectsAndClosures.remove(null);  
364 final int size = objectsAndClosures.size();  
365 final Closure<? super E>[] trs = new Closure[size];  
366 final Predicate<E>[] preds = new Predicate[size];  
367 int i = 0;  
368 for (final Map.Entry<? extends E, Closure<E>> entry : objectsAndClosures.entrySet()) {  
369 preds[i] = EqualPredicate.<E>equalPredicate(entry.getKey());  
370 trs[i] = entry.getValue();  
371 i++;  
372 }  
373 return ClosureUtils.<E>switchClosure(preds, trs, def);  
374 }  
375  
376}