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.Closure;  
023import org.apache.commons.collections4.Predicate;  
024  
025/\*\*  
026 \* Closure implementation calls the closure whose predicate returns true,  
027 \* like a switch statement.  
028 \*  
029 \* @since 3.0  
030 \*/  
031public class SwitchClosure<E> implements Closure<E>, Serializable {  
032  
033 /\*\* Serial version UID \*/  
034 private static final long serialVersionUID = 3518477308466486130L;  
035  
036 /\*\* The tests to consider \*/  
037 private final Predicate<? super E>[] iPredicates;  
038 /\*\* The matching closures to call \*/  
039 private final Closure<? super E>[] iClosures;  
040 /\*\* The default closure to call if no tests match \*/  
041 private final Closure<? super E> iDefault;  
042  
043 /\*\*  
044 \* Factory method that performs validation and copies the parameter arrays.  
045 \*  
046 \* @param <E> the type that the closure acts on  
047 \* @param predicates array of predicates, cloned, no nulls  
048 \* @param closures matching array of closures, cloned, no nulls  
049 \* @param defaultClosure the closure to use if no match, null means nop  
050 \* @return the <code>chained</code> closure  
051 \* @throws NullPointerException if array is null  
052 \* @throws NullPointerException if any element in the array is null  
053 \* @throws IllegalArgumentException if the array lengths of predicates and closures do not match  
054 \*/  
055 @SuppressWarnings("unchecked")  
056 public static <E> Closure<E> switchClosure(final Predicate<? super E>[] predicates,  
057 final Closure<? super E>[] closures,  
058 final Closure<? super E> defaultClosure) {  
059 FunctorUtils.validate(predicates);  
060 FunctorUtils.validate(closures);  
061 if (predicates.length != closures.length) {  
062 throw new IllegalArgumentException("The predicate and closure arrays must be the same size");  
063 }  
064 if (predicates.length == 0) {  
065 return (Closure<E>) (defaultClosure == null ? NOPClosure.<E>nopClosure(): defaultClosure);  
066 }  
067 return new SwitchClosure<>(predicates, closures, defaultClosure);  
068 }  
069  
070 /\*\*  
071 \* Create a new Closure that calls one of the closures depending  
072 \* on the predicates.  
073 \* <p>  
074 \* The Map consists of Predicate keys and Closure values. A closure  
075 \* is called if its matching predicate returns true. Each predicate is evaluated  
076 \* until one returns true. If no predicates evaluate to true, the default  
077 \* closure is called. The default closure is set in the map with a  
078 \* null key. The ordering is that of the iterator() method on the entryset  
079 \* collection of the map.  
080 \*  
081 \* @param <E> the type that the closure acts on  
082 \* @param predicatesAndClosures a map of predicates to closures  
083 \* @return the <code>switch</code> closure  
084 \* @throws NullPointerException if the map is null  
085 \* @throws NullPointerException if any closure in the map is null  
086 \* @throws ClassCastException if the map elements are of the wrong type  
087 \*/  
088 @SuppressWarnings("unchecked")  
089 public static <E> Closure<E> switchClosure(final Map<Predicate<E>, Closure<E>> predicatesAndClosures) {  
090 if (predicatesAndClosures == null) {  
091 throw new NullPointerException("The predicate and closure map must not be null");  
092 }  
093 // convert to array like this to guarantee iterator() ordering  
094 final Closure<? super E> defaultClosure = predicatesAndClosures.remove(null);  
095 final int size = predicatesAndClosures.size();  
096 if (size == 0) {  
097 return (Closure<E>) (defaultClosure == null ? NOPClosure.<E>nopClosure() : defaultClosure);  
098 }  
099 final Closure<E>[] closures = new Closure[size];  
100 final Predicate<E>[] preds = new Predicate[size];  
101 int i = 0;  
102 for (final Map.Entry<Predicate<E>, Closure<E>> entry : predicatesAndClosures.entrySet()) {  
103 preds[i] = entry.getKey();  
104 closures[i] = entry.getValue();  
105 i++;  
106 }  
107 return new SwitchClosure<>(false, preds, closures, defaultClosure);  
108 }  
109  
110 /\*\*  
111 \* Hidden constructor for the use by the static factory methods.  
112 \*  
113 \* @param clone if {@code true} the input arguments will be cloned  
114 \* @param predicates array of predicates, no nulls  
115 \* @param closures matching array of closures, no nulls  
116 \* @param defaultClosure the closure to use if no match, null means nop  
117 \*/  
118 @SuppressWarnings("unchecked")  
119 private SwitchClosure(final boolean clone, final Predicate<? super E>[] predicates,  
120 final Closure<? super E>[] closures, final Closure<? super E> defaultClosure) {  
121 super();  
122 iPredicates = clone ? FunctorUtils.copy(predicates) : predicates;  
123 iClosures = clone ? FunctorUtils.copy(closures) : closures;  
124 iDefault = (Closure<? super E>) (defaultClosure == null ? NOPClosure.<E>nopClosure() : defaultClosure);  
125 }  
126  
127 /\*\*  
128 \* Constructor that performs no validation.  
129 \* Use <code>switchClosure</code> if you want that.  
130 \*  
131 \* @param predicates array of predicates, cloned, no nulls  
132 \* @param closures matching array of closures, cloned, no nulls  
133 \* @param defaultClosure the closure to use if no match, null means nop  
134 \*/  
135 public SwitchClosure(final Predicate<? super E>[] predicates, final Closure<? super E>[] closures,  
136 final Closure<? super E> defaultClosure) {  
137 this(true, predicates, closures, defaultClosure);  
138 }  
139  
140 /\*\*  
141 \* Executes the closure whose matching predicate returns true  
142 \*  
143 \* @param input the input object  
144 \*/  
145 @Override  
146 public void execute(final E input) {  
147 for (int i = 0; i < iPredicates.length; i++) {  
148 if (iPredicates[i].evaluate(input) == true) {  
149 iClosures[i].execute(input);  
150 return;  
151 }  
152 }  
153 iDefault.execute(input);  
154 }  
155  
156 /\*\*  
157 \* Gets the predicates.  
158 \*  
159 \* @return a copy of the predicates  
160 \* @since 3.1  
161 \*/  
162 public Predicate<? super E>[] getPredicates() {  
163 return FunctorUtils.<E>copy(iPredicates);  
164 }  
165  
166 /\*\*  
167 \* Gets the closures.  
168 \*  
169 \* @return a copy of the closures  
170 \* @since 3.1  
171 \*/  
172 public Closure<? super E>[] getClosures() {  
173 return FunctorUtils.<E>copy(iClosures);  
174 }  
175  
176 /\*\*  
177 \* Gets the default closure.  
178 \*  
179 \* @return the default closure  
180 \* @since 3.1  
181 \*/  
182 public Closure<? super E> getDefaultClosure() {  
183 return iDefault;  
184 }  
185  
186}