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.Collection;  
021  
022import org.apache.commons.collections4.Closure;  
023  
024/\*\*  
025 \* Closure implementation that chains the specified closures together.  
026 \*  
027 \* @since 3.0  
028 \*/  
029public class ChainedClosure<E> implements Closure<E>, Serializable {  
030  
031 /\*\* Serial version UID \*/  
032 private static final long serialVersionUID = -3520677225766901240L;  
033  
034 /\*\* The closures to call in turn \*/  
035 private final Closure<? super E>[] iClosures;  
036  
037 /\*\*  
038 \* Factory method that performs validation and copies the parameter array.  
039 \*  
040 \* @param <E> the type that the closure acts on  
041 \* @param closures the closures to chain, copied, no nulls  
042 \* @return the <code>chained</code> closure  
043 \* @throws NullPointerException if the closures array is null  
044 \* @throws NullPointerException if any closure in the array is null  
045 \*/  
046 public static <E> Closure<E> chainedClosure(final Closure<? super E>... closures) {  
047 FunctorUtils.validate(closures);  
048 if (closures.length == 0) {  
049 return NOPClosure.<E>nopClosure();  
050 }  
051 return new ChainedClosure<>(closures);  
052 }  
053  
054 /\*\*  
055 \* Create a new Closure that calls each closure in turn, passing the  
056 \* result into the next closure. The ordering is that of the iterator()  
057 \* method on the collection.  
058 \*  
059 \* @param <E> the type that the closure acts on  
060 \* @param closures a collection of closures to chain  
061 \* @return the <code>chained</code> closure  
062 \* @throws NullPointerException if the closures collection is null  
063 \* @throws NullPointerException if any closure in the collection is null  
064 \*/  
065 @SuppressWarnings("unchecked")  
066 public static <E> Closure<E> chainedClosure(final Collection<? extends Closure<? super E>> closures) {  
067 if (closures == null) {  
068 throw new NullPointerException("Closure collection must not be null");  
069 }  
070 if (closures.size() == 0) {  
071 return NOPClosure.<E>nopClosure();  
072 }  
073 // convert to array like this to guarantee iterator() ordering  
074 final Closure<? super E>[] cmds = new Closure[closures.size()];  
075 int i = 0;  
076 for (final Closure<? super E> closure : closures) {  
077 cmds[i++] = closure;  
078 }  
079 FunctorUtils.validate(cmds);  
080 return new ChainedClosure<>(false, cmds);  
081 }  
082  
083 /\*\*  
084 \* Hidden constructor for the use by the static factory methods.  
085 \*  
086 \* @param clone if {@code true} the input argument will be cloned  
087 \* @param closures the closures to chain, no nulls  
088 \*/  
089 private ChainedClosure(final boolean clone, final Closure<? super E>... closures) {  
090 super();  
091 iClosures = clone ? FunctorUtils.copy(closures) : closures;  
092 }  
093  
094 /\*\*  
095 \* Constructor that performs no validation.  
096 \* Use <code>chainedClosure</code> if you want that.  
097 \*  
098 \* @param closures the closures to chain, copied, no nulls  
099 \*/  
100 public ChainedClosure(final Closure<? super E>... closures) {  
101 this(true, closures);  
102 }  
103  
104 /\*\*  
105 \* Execute a list of closures.  
106 \*  
107 \* @param input the input object passed to each closure  
108 \*/  
109 @Override  
110 public void execute(final E input) {  
111 for (final Closure<? super E> iClosure : iClosures) {  
112 iClosure.execute(input);  
113 }  
114 }  
115  
116 /\*\*  
117 \* Gets the closures.  
118 \*  
119 \* @return a copy of the closures  
120 \* @since 3.1  
121 \*/  
122 public Closure<? super E>[] getClosures() {  
123 return FunctorUtils.<E>copy(iClosures);  
124 }  
125  
126}