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.Transformer;  
023  
024/\*\*  
025 \* Transformer implementation that chains the specified transformers together.  
026 \* <p>  
027 \* The input object is passed to the first transformer. The transformed result  
028 \* is passed to the second transformer and so on.  
029 \* </p>  
030 \*  
031 \* @since 3.0  
032 \*/  
033public class ChainedTransformer<T> implements Transformer<T, T>, Serializable {  
034  
035 /\*\* Serial version UID \*/  
036 private static final long serialVersionUID = 3514945074733160196L;  
037  
038 /\*\* The transformers to call in turn \*/  
039 private final Transformer<? super T, ? extends T>[] iTransformers;  
040  
041 /\*\*  
042 \* Factory method that performs validation and copies the parameter array.  
043 \*  
044 \* @param <T> the object type  
045 \* @param transformers the transformers to chain, copied, no nulls  
046 \* @return the <code>chained</code> transformer  
047 \* @throws NullPointerException if the transformers array is null  
048 \* @throws NullPointerException if any transformer in the array is null  
049 \*/  
050 public static <T> Transformer<T, T> chainedTransformer(final Transformer<? super T, ? extends T>... transformers) {  
051 FunctorUtils.validate(transformers);  
052 if (transformers.length == 0) {  
053 return NOPTransformer.<T>nopTransformer();  
054 }  
055 return new ChainedTransformer<>(transformers);  
056 }  
057  
058 /\*\*  
059 \* Create a new Transformer that calls each transformer in turn, passing the  
060 \* result into the next transformer. The ordering is that of the iterator()  
061 \* method on the collection.  
062 \*  
063 \* @param <T> the object type  
064 \* @param transformers a collection of transformers to chain  
065 \* @return the <code>chained</code> transformer  
066 \* @throws NullPointerException if the transformers collection is null  
067 \* @throws NullPointerException if any transformer in the collection is null  
068 \*/  
069 public static <T> Transformer<T, T> chainedTransformer(  
070 final Collection<? extends Transformer<? super T, ? extends T>> transformers) {  
071 if (transformers == null) {  
072 throw new NullPointerException("Transformer collection must not be null");  
073 }  
074 if (transformers.size() == 0) {  
075 return NOPTransformer.<T>nopTransformer();  
076 }  
077 // convert to array like this to guarantee iterator() ordering  
078 final Transformer<T, T>[] cmds = transformers.toArray(new Transformer[transformers.size()]);  
079 FunctorUtils.validate(cmds);  
080 return new ChainedTransformer<>(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 transformers the transformers to chain, no nulls  
088 \*/  
089 private ChainedTransformer(final boolean clone, final Transformer<? super T, ? extends T>[] transformers) {  
090 super();  
091 iTransformers = clone ? FunctorUtils.copy(transformers) : transformers;  
092 }  
093  
094 /\*\*  
095 \* Constructor that performs no validation.  
096 \* Use <code>chainedTransformer</code> if you want that.  
097 \*  
098 \* @param transformers the transformers to chain, copied, no nulls  
099 \*/  
100 public ChainedTransformer(final Transformer<? super T, ? extends T>... transformers) {  
101 this(true, transformers);  
102 }  
103  
104 /\*\*  
105 \* Transforms the input to result via each decorated transformer  
106 \*  
107 \* @param object the input object passed to the first transformer  
108 \* @return the transformed result  
109 \*/  
110 @Override  
111 public T transform(T object) {  
112 for (final Transformer<? super T, ? extends T> iTransformer : iTransformers) {  
113 object = iTransformer.transform(object);  
114 }  
115 return object;  
116 }  
117  
118 /\*\*  
119 \* Gets the transformers.  
120 \*  
121 \* @return a copy of the transformers  
122 \* @since 3.1  
123 \*/  
124 public Transformer<? super T, ? extends T>[] getTransformers() {  
125 return FunctorUtils.<T, T>copy(iTransformers);  
126 }  
127  
128}