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;  
020  
021import org.apache.commons.collections4.Closure;  
022import org.apache.commons.collections4.Transformer;  
023  
024/\*\*  
025 \* Transformer implementation that calls a Closure using the input object  
026 \* and then returns the input.  
027 \*  
028 \* @since 3.0  
029 \*/  
030public class ClosureTransformer<T> implements Transformer<T, T>, Serializable {  
031  
032 /\*\* Serial version UID \*/  
033 private static final long serialVersionUID = 478466901448617286L;  
034  
035 /\*\* The closure to wrap \*/  
036 private final Closure<? super T> iClosure;  
037  
038 /\*\*  
039 \* Factory method that performs validation.  
040 \*  
041 \* @param <T> the type of the object to transform  
042 \* @param closure the closure to call, not null  
043 \* @return the <code>closure</code> transformer  
044 \* @throws NullPointerException if the closure is null  
045 \*/  
046 public static <T> Transformer<T, T> closureTransformer(final Closure<? super T> closure) {  
047 if (closure == null) {  
048 throw new NullPointerException("Closure must not be null");  
049 }  
050 return new ClosureTransformer<>(closure);  
051 }  
052  
053 /\*\*  
054 \* Constructor that performs no validation.  
055 \* Use <code>closureTransformer</code> if you want that.  
056 \*  
057 \* @param closure the closure to call, not null  
058 \*/  
059 public ClosureTransformer(final Closure<? super T> closure) {  
060 super();  
061 iClosure = closure;  
062 }  
063  
064 /\*\*  
065 \* Transforms the input to result by executing a closure.  
066 \*  
067 \* @param input the input object to transform  
068 \* @return the transformed result  
069 \*/  
070 @Override  
071 public T transform(final T input) {  
072 iClosure.execute(input);  
073 return input;  
074 }  
075  
076 /\*\*  
077 \* Gets the closure.  
078 \*  
079 \* @return the closure  
080 \* @since 3.1  
081 \*/  
082 public Closure<? super T> getClosure() {  
083 return iClosure;  
084 }  
085  
086}