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 org.apache.commons.collections4.Closure;  
020  
021/\*\*  
022 \* Closure implementation that calls another closure n times, like a for loop.  
023 \* <p>  
024 \* <b>WARNING:</b> from v4.1 onwards this class will <b>not</b> be serializable anymore  
025 \* in order to prevent potential remote code execution exploits. Please refer to  
026 \* <a href="https://issues.apache.org/jira/browse/COLLECTIONS-580">COLLECTIONS-580</a>  
027 \* for more details.  
028 \* </p>  
029 \*  
030 \* @since 3.0  
031 \*/  
032public class ForClosure<E> implements Closure<E> {  
033  
034 /\*\* The number of times to loop \*/  
035 private final int iCount;  
036 /\*\* The closure to call \*/  
037 private final Closure<? super E> iClosure;  
038  
039 /\*\*  
040 \* Factory method that performs validation.  
041 \* <p>  
042 \* A null closure or zero count returns the <code>NOPClosure</code>.  
043 \* A count of one returns the specified closure.  
044 \*  
045 \* @param <E> the type that the closure acts on  
046 \* @param count the number of times to execute the closure  
047 \* @param closure the closure to execute, not null  
048 \* @return the <code>for</code> closure  
049 \*/  
050 @SuppressWarnings("unchecked")  
051 public static <E> Closure<E> forClosure(final int count, final Closure<? super E> closure) {  
052 if (count <= 0 || closure == null) {  
053 return NOPClosure.<E>nopClosure();  
054 }  
055 if (count == 1) {  
056 return (Closure<E>) closure;  
057 }  
058 return new ForClosure<>(count, closure);  
059 }  
060  
061 /\*\*  
062 \* Constructor that performs no validation.  
063 \* Use <code>forClosure</code> if you want that.  
064 \*  
065 \* @param count the number of times to execute the closure  
066 \* @param closure the closure to execute, not null  
067 \*/  
068 public ForClosure(final int count, final Closure<? super E> closure) {  
069 super();  
070 iCount = count;  
071 iClosure = closure;  
072 }  
073  
074 /\*\*  
075 \* Executes the closure <code>count</code> times.  
076 \*  
077 \* @param input the input object  
078 \*/  
079 @Override  
080 public void execute(final E input) {  
081 for (int i = 0; i < iCount; i++) {  
082 iClosure.execute(input);  
083 }  
084 }  
085  
086 /\*\*  
087 \* Gets the closure.  
088 \*  
089 \* @return the closure  
090 \* @since 3.1  
091 \*/  
092 public Closure<? super E> getClosure() {  
093 return iClosure;  
094 }  
095  
096 /\*\*  
097 \* Gets the count.  
098 \*  
099 \* @return the count  
100 \* @since 3.1  
101 \*/  
102 public int getCount() {  
103 return iCount;  
104 }  
105  
106}