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;  
020import org.apache.commons.collections4.Predicate;  
021  
022/\*\*  
023 \* Closure implementation that executes a closure repeatedly until a condition is met,  
024 \* like a do-while or while loop.  
025 \* <p>  
026 \* <b>WARNING:</b> from v4.1 onwards this class will <b>not</b> be serializable anymore  
027 \* in order to prevent potential remote code execution exploits. Please refer to  
028 \* <a href="https://issues.apache.org/jira/browse/COLLECTIONS-580">COLLECTIONS-580</a>  
029 \* for more details.  
030 \* </p>  
031 \*  
032 \* @since 3.0  
033 \*/  
034public class WhileClosure<E> implements Closure<E> {  
035  
036 /\*\* The test condition \*/  
037 private final Predicate<? super E> iPredicate;  
038 /\*\* The closure to call \*/  
039 private final Closure<? super E> iClosure;  
040 /\*\* The flag, true is a do loop, false is a while \*/  
041 private final boolean iDoLoop;  
042  
043 /\*\*  
044 \* Factory method that performs validation.  
045 \*  
046 \* @param <E> the type that the closure acts on  
047 \* @param predicate the predicate used to evaluate when the loop terminates, not null  
048 \* @param closure the closure the execute, not null  
049 \* @param doLoop true to act as a do-while loop, always executing the closure once  
050 \* @return the <code>while</code> closure  
051 \* @throws NullPointerException if the predicate or closure is null  
052 \*/  
053 public static <E> Closure<E> whileClosure(final Predicate<? super E> predicate,  
054 final Closure<? super E> closure, final boolean doLoop) {  
055 if (predicate == null) {  
056 throw new NullPointerException("Predicate must not be null");  
057 }  
058 if (closure == null) {  
059 throw new NullPointerException("Closure must not be null");  
060 }  
061 return new WhileClosure<>(predicate, closure, doLoop);  
062 }  
063  
064 /\*\*  
065 \* Constructor that performs no validation.  
066 \* Use <code>whileClosure</code> if you want that.  
067 \*  
068 \* @param predicate the predicate used to evaluate when the loop terminates, not null  
069 \* @param closure the closure the execute, not null  
070 \* @param doLoop true to act as a do-while loop, always executing the closure once  
071 \*/  
072 public WhileClosure(final Predicate<? super E> predicate, final Closure<? super E> closure, final boolean doLoop) {  
073 super();  
074 iPredicate = predicate;  
075 iClosure = closure;  
076 iDoLoop = doLoop;  
077 }  
078  
079 /\*\*  
080 \* Executes the closure until the predicate is false.  
081 \*  
082 \* @param input the input object  
083 \*/  
084 @Override  
085 public void execute(final E input) {  
086 if (iDoLoop) {  
087 iClosure.execute(input);  
088 }  
089 while (iPredicate.evaluate(input)) {  
090 iClosure.execute(input);  
091 }  
092 }  
093  
094 /\*\*  
095 \* Gets the predicate in use.  
096 \*  
097 \* @return the predicate  
098 \* @since 3.1  
099 \*/  
100 public Predicate<? super E> getPredicate() {  
101 return iPredicate;  
102 }  
103  
104 /\*\*  
105 \* Gets the closure.  
106 \*  
107 \* @return the closure  
108 \* @since 3.1  
109 \*/  
110 public Closure<? super E> getClosure() {  
111 return iClosure;  
112 }  
113  
114 /\*\*  
115 \* Is the loop a do-while loop.  
116 \*  
117 \* @return true is do-while, false if while  
118 \* @since 3.1  
119 \*/  
120 public boolean isDoLoop() {  
121 return iDoLoop;  
122 }  
123  
124}