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.lang.reflect.InvocationTargetException;  
020import java.lang.reflect.Method;  
021  
022import org.apache.commons.collections4.FunctorException;  
023import org.apache.commons.collections4.Transformer;  
024  
025/\*\*  
026 \* Transformer implementation that creates a new object instance by reflection.  
027 \* <p>  
028 \* <b>WARNING:</b> from v4.1 onwards this class will <b>not</b> be serializable anymore  
029 \* in order to prevent potential remote code execution exploits. Please refer to  
030 \* <a href="https://issues.apache.org/jira/browse/COLLECTIONS-580">COLLECTIONS-580</a>  
031 \* for more details.  
032 \* </p>  
033 \*  
034 \* @since 3.0  
035 \*/  
036public class InvokerTransformer<I, O> implements Transformer<I, O> {  
037  
038 /\*\* The method name to call \*/  
039 private final String iMethodName;  
040 /\*\* The array of reflection parameter types \*/  
041 private final Class<?>[] iParamTypes;  
042 /\*\* The array of reflection arguments \*/  
043 private final Object[] iArgs;  
044  
045 /\*\*  
046 \* Gets an instance of this transformer calling a specific method with no arguments.  
047 \*  
048 \* @param <I> the input type  
049 \* @param <O> the output type  
050 \* @param methodName the method name to call  
051 \* @return an invoker transformer  
052 \* @throws NullPointerException if methodName is null  
053 \* @since 3.1  
054 \*/  
055 public static <I, O> Transformer<I, O> invokerTransformer(final String methodName) {  
056 if (methodName == null) {  
057 throw new NullPointerException("The method to invoke must not be null");  
058 }  
059 return new InvokerTransformer<>(methodName);  
060 }  
061  
062 /\*\*  
063 \* Gets an instance of this transformer calling a specific method with specific values.  
064 \*  
065 \* @param <I> the input type  
066 \* @param <O> the output type  
067 \* @param methodName the method name to call  
068 \* @param paramTypes the parameter types of the method  
069 \* @param args the arguments to pass to the method  
070 \* @return an invoker transformer  
071 \* @throws NullPointerException if methodName is null  
072 \* @throws IllegalArgumentException if paramTypes does not match args  
073 \*/  
074 public static <I, O> Transformer<I, O> invokerTransformer(final String methodName, final Class<?>[] paramTypes,  
075 final Object[] args) {  
076 if (methodName == null) {  
077 throw new NullPointerException("The method to invoke must not be null");  
078 }  
079 if (((paramTypes == null) && (args != null))  
080 || ((paramTypes != null) && (args == null))  
081 || ((paramTypes != null) && (args != null) && (paramTypes.length != args.length))) {  
082 throw new IllegalArgumentException("The parameter types must match the arguments");  
083 }  
084 if (paramTypes == null || paramTypes.length == 0) {  
085 return new InvokerTransformer<>(methodName);  
086 }  
087 return new InvokerTransformer<>(methodName, paramTypes, args);  
088 }  
089  
090 /\*\*  
091 \* Constructor for no arg instance.  
092 \*  
093 \* @param methodName the method to call  
094 \*/  
095 private InvokerTransformer(final String methodName) {  
096 super();  
097 iMethodName = methodName;  
098 iParamTypes = null;  
099 iArgs = null;  
100 }  
101  
102 /\*\*  
103 \* Constructor that performs no validation.  
104 \* Use <code>invokerTransformer</code> if you want that.  
105 \* <p>  
106 \* Note: from 4.0, the input parameters will be cloned  
107 \*  
108 \* @param methodName the method to call  
109 \* @param paramTypes the constructor parameter types  
110 \* @param args the constructor arguments  
111 \*/  
112 public InvokerTransformer(final String methodName, final Class<?>[] paramTypes, final Object[] args) {  
113 super();  
114 iMethodName = methodName;  
115 iParamTypes = paramTypes != null ? paramTypes.clone() : null;  
116 iArgs = args != null ? args.clone() : null;  
117 }  
118  
119 /\*\*  
120 \* Transforms the input to result by invoking a method on the input.  
121 \*  
122 \* @param input the input object to transform  
123 \* @return the transformed result, null if null input  
124 \*/  
125 @Override  
126 @SuppressWarnings("unchecked")  
127 public O transform(final Object input) {  
128 if (input == null) {  
129 return null;  
130 }  
131 try {  
132 final Class<?> cls = input.getClass();  
133 final Method method = cls.getMethod(iMethodName, iParamTypes);  
134 return (O) method.invoke(input, iArgs);  
135 } catch (final NoSuchMethodException ex) {  
136 throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" +  
137 input.getClass() + "' does not exist");  
138 } catch (final IllegalAccessException ex) {  
139 throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" +  
140 input.getClass() + "' cannot be accessed");  
141 } catch (final InvocationTargetException ex) {  
142 throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" +  
143 input.getClass() + "' threw an exception", ex);  
144 }  
145 }  
146  
147}