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.Constructor;  
020import java.lang.reflect.InvocationTargetException;  
021  
022import org.apache.commons.collections4.Factory;  
023import org.apache.commons.collections4.FunctorException;  
024  
025/\*\*  
026 \* Factory 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 InstantiateFactory<T> implements Factory<T> {  
037  
038 /\*\* The class to create \*/  
039 private final Class<T> iClassToInstantiate;  
040 /\*\* The constructor parameter types \*/  
041 private final Class<?>[] iParamTypes;  
042 /\*\* The constructor arguments \*/  
043 private final Object[] iArgs;  
044 /\*\* The constructor \*/  
045 private transient Constructor<T> iConstructor = null;  
046  
047 /\*\*  
048 \* Factory method that performs validation.  
049 \*  
050 \* @param <T> the type the factory creates  
051 \* @param classToInstantiate the class to instantiate, not null  
052 \* @param paramTypes the constructor parameter types, cloned  
053 \* @param args the constructor arguments, cloned  
054 \* @return a new instantiate factory  
055 \* @throws NullPointerException if classToInstantiate is null  
056 \* @throws IllegalArgumentException if paramTypes does not match args  
057 \*/  
058 public static <T> Factory<T> instantiateFactory(final Class<T> classToInstantiate,  
059 final Class<?>[] paramTypes,  
060 final Object[] args) {  
061 if (classToInstantiate == null) {  
062 throw new NullPointerException("Class to instantiate must not be null");  
063 }  
064 if (paramTypes == null && args != null  
065 || paramTypes != null && args == null  
066 || paramTypes != null && args != null && paramTypes.length != args.length) {  
067 throw new IllegalArgumentException("Parameter types must match the arguments");  
068 }  
069  
070 if (paramTypes == null || paramTypes.length == 0) {  
071 return new InstantiateFactory<>(classToInstantiate);  
072 }  
073 return new InstantiateFactory<>(classToInstantiate, paramTypes, args);  
074 }  
075  
076 /\*\*  
077 \* Constructor that performs no validation.  
078 \* Use <code>instantiateFactory</code> if you want that.  
079 \*  
080 \* @param classToInstantiate the class to instantiate  
081 \*/  
082 public InstantiateFactory(final Class<T> classToInstantiate) {  
083 super();  
084 iClassToInstantiate = classToInstantiate;  
085 iParamTypes = null;  
086 iArgs = null;  
087 findConstructor();  
088 }  
089  
090 /\*\*  
091 \* Constructor that performs no validation.  
092 \* Use <code>instantiateFactory</code> if you want that.  
093 \*  
094 \* @param classToInstantiate the class to instantiate  
095 \* @param paramTypes the constructor parameter types, cloned  
096 \* @param args the constructor arguments, cloned  
097 \*/  
098 public InstantiateFactory(final Class<T> classToInstantiate, final Class<?>[] paramTypes, final Object[] args) {  
099 super();  
100 iClassToInstantiate = classToInstantiate;  
101 iParamTypes = paramTypes.clone();  
102 iArgs = args.clone();  
103 findConstructor();  
104 }  
105  
106 /\*\*  
107 \* Find the Constructor for the class specified.  
108 \*/  
109 private void findConstructor() {  
110 try {  
111 iConstructor = iClassToInstantiate.getConstructor(iParamTypes);  
112 } catch (final NoSuchMethodException ex) {  
113 throw new IllegalArgumentException("InstantiateFactory: The constructor must exist and be public ");  
114 }  
115 }  
116  
117 /\*\*  
118 \* Creates an object using the stored constructor.  
119 \*  
120 \* @return the new object  
121 \*/  
122 @Override  
123 public T create() {  
124 // needed for post-serialization  
125 if (iConstructor == null) {  
126 findConstructor();  
127 }  
128  
129 try {  
130 return iConstructor.newInstance(iArgs);  
131 } catch (final InstantiationException ex) {  
132 throw new FunctorException("InstantiateFactory: InstantiationException", ex);  
133 } catch (final IllegalAccessException ex) {  
134 throw new FunctorException("InstantiateFactory: Constructor must be public", ex);  
135 } catch (final InvocationTargetException ex) {  
136 throw new FunctorException("InstantiateFactory: Constructor threw an exception", ex);  
137 }  
138 }  
139  
140}