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;  
018  
019import org.apache.commons.collections4.functors.ConstantFactory;  
020import org.apache.commons.collections4.functors.ExceptionFactory;  
021import org.apache.commons.collections4.functors.InstantiateFactory;  
022import org.apache.commons.collections4.functors.PrototypeFactory;  
023  
024/\*\*  
025 \* <code>FactoryUtils</code> provides reference implementations and utilities  
026 \* for the Factory functor interface. The supplied factories are:  
027 \* <ul>  
028 \* <li>Prototype - clones a specified object  
029 \* <li>Instantiate - creates objects using reflection  
030 \* <li>Constant - always returns the same object  
031 \* <li>Null - always returns null  
032 \* <li>Exception - always throws an exception  
033 \* </ul>  
034 \* <p>  
035 \* Since v4.1 only factories which are considered to be safe are  
036 \* Serializable. Factories considered to be unsafe for serialization are:  
037 \* <ul>  
038 \* <li>Prototype  
039 \* <li>Instantiate  
040 \* </ul>  
041 \*  
042 \* @since 3.0  
043 \*/  
044public class FactoryUtils {  
045  
046 /\*\*  
047 \* This class is not normally instantiated.  
048 \*/  
049 private FactoryUtils() {}  
050  
051 /\*\*  
052 \* Gets a Factory that always throws an exception.  
053 \* This could be useful during testing as a placeholder.  
054 \*  
055 \* @see org.apache.commons.collections4.functors.ExceptionFactory  
056 \*  
057 \* @param <T> the type that the factory creates  
058 \* @return the factory  
059 \*/  
060 public static <T> Factory<T> exceptionFactory() {  
061 return ExceptionFactory.<T>exceptionFactory();  
062 }  
063  
064 /\*\*  
065 \* Gets a Factory that will return null each time the factory is used.  
066 \* This could be useful during testing as a placeholder.  
067 \*  
068 \* @see org.apache.commons.collections4.functors.ConstantFactory  
069 \* @param <T> the "type" of null object the factory should return.  
070 \* @return the factory  
071 \*/  
072 public static <T> Factory<T> nullFactory() {  
073 return ConstantFactory.<T>constantFactory(null);  
074 }  
075  
076 /\*\*  
077 \* Creates a Factory that will return the same object each time the factory  
078 \* is used. No check is made that the object is immutable. In general, only  
079 \* immutable objects should use the constant factory. Mutable objects should  
080 \* use the prototype factory.  
081 \*  
082 \* @see org.apache.commons.collections4.functors.ConstantFactory  
083 \*  
084 \* @param <T> the type that the factory creates  
085 \* @param constantToReturn the constant object to return each time in the factory  
086 \* @return the <code>constant</code> factory.  
087 \*/  
088 public static <T> Factory<T> constantFactory(final T constantToReturn) {  
089 return ConstantFactory.constantFactory(constantToReturn);  
090 }  
091  
092 /\*\*  
093 \* Creates a Factory that will return a clone of the same prototype object  
094 \* each time the factory is used. The prototype will be cloned using one of these  
095 \* techniques (in order):  
096 \*  
097 \* <ul>  
098 \* <li>public clone method</li>  
099 \* <li>public copy constructor</li>  
100 \* <li>serialization clone</li>  
101 \* </ul>  
102 \*  
103 \* @see org.apache.commons.collections4.functors.PrototypeFactory  
104 \*  
105 \* @param <T> the type that the factory creates  
106 \* @param prototype the object to clone each time in the factory  
107 \* @return the <code>prototype</code> factory, or a {@link ConstantFactory#NULL\_INSTANCE} if  
108 \* the {@code prototype} is {@code null}  
109 \* @throws IllegalArgumentException if the prototype cannot be cloned  
110 \*/  
111 public static <T> Factory<T> prototypeFactory(final T prototype) {  
112 return PrototypeFactory.<T>prototypeFactory(prototype);  
113 }  
114  
115 /\*\*  
116 \* Creates a Factory that can create objects of a specific type using  
117 \* a no-args constructor.  
118 \*  
119 \* @see org.apache.commons.collections4.functors.InstantiateFactory  
120 \*  
121 \* @param <T> the type that the factory creates  
122 \* @param classToInstantiate the Class to instantiate each time in the factory  
123 \* @return the <code>reflection</code> factory  
124 \* @throws NullPointerException if the classToInstantiate is null  
125 \*/  
126 public static <T> Factory<T> instantiateFactory(final Class<T> classToInstantiate) {  
127 return InstantiateFactory.instantiateFactory(classToInstantiate, null, null);  
128 }  
129  
130 /\*\*  
131 \* Creates a Factory that can create objects of a specific type using  
132 \* the arguments specified to this method.  
133 \*  
134 \* @see org.apache.commons.collections4.functors.InstantiateFactory  
135 \*  
136 \* @param <T> the type that the factory creates  
137 \* @param classToInstantiate the Class to instantiate each time in the factory  
138 \* @param paramTypes parameter types for the constructor, can be null  
139 \* @param args the arguments to pass to the constructor, can be null  
140 \* @return the <code>reflection</code> factory  
141 \* @throws NullPointerException if the classToInstantiate is null  
142 \* @throws IllegalArgumentException if the paramTypes and args don't match  
143 \* @throws IllegalArgumentException if the constructor doesn't exist  
144 \*/  
145 public static <T> Factory<T> instantiateFactory(final Class<T> classToInstantiate, final Class<?>[] paramTypes,  
146 final Object[] args) {  
147 return InstantiateFactory.instantiateFactory(classToInstantiate, paramTypes, args);  
148 }  
149  
150}