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.beanutils;  
018  
019import java.beans.IntrospectionException;  
020import java.beans.Introspector;  
021import java.beans.PropertyDescriptor;  
022import java.lang.reflect.Method;  
023import java.util.Locale;  
024  
025import org.apache.commons.logging.Log;  
026import org.apache.commons.logging.LogFactory;  
027  
028/\*\*  
029 \* <p>  
030 \* An implementation of the <code>BeanIntrospector</code> interface which can  
031 \* detect write methods for properties used in fluent API scenario.  
032 \* </p>  
033 \* <p>  
034 \* A <em>fluent API</em> allows setting multiple properties using a single  
035 \* statement by supporting so-called <em>method chaining</em>: Methods for  
036 \* setting a property value do not return <b>void</b>, but an object which can  
037 \* be called for setting another property. An example of such a fluent API could  
038 \* look as follows:  
039 \*  
040 \* <pre>  
041 \* public class FooBuilder {  
042 \* public FooBuilder setFooProperty1(String value) {  
043 \* ...  
044 \* return this;  
045 \* }  
046 \*  
047 \* public FooBuilder setFooProperty2(int value) {  
048 \* ...  
049 \* return this;  
050 \* }  
051 \* }  
052 \* </pre>  
053 \*  
054 \* Per default, <code>PropertyUtils</code> does not detect methods like this  
055 \* because, having a non-<b>void</b> return type, they violate the Java Beans  
056 \* specification.  
057 \* </p>  
058 \* <p>  
059 \* This class is more tolerant with regards to the return type of a set method.  
060 \* It basically iterates over all methods of a class and filters them for a  
061 \* configurable prefix (the default prefix is <code>set</code>). It then  
062 \* generates corresponding <code>PropertyDescriptor</code> objects for the  
063 \* methods found which use these methods as write methods.  
064 \* </p>  
065 \* <p>  
066 \* An instance of this class is intended to collaborate with a  
067 \* {@link DefaultBeanIntrospector} object. So best results are achieved by  
068 \* adding this instance as custom {@code BeanIntrospector} after the  
069 \* <code>DefaultBeanIntrospector</code> object. Then default introspection finds  
070 \* read-only properties because it does not detect the write methods with a  
071 \* non-<b>void</b> return type. {@code FluentPropertyBeanIntrospector}  
072 \* completes the descriptors for these properties by setting the correct write  
073 \* method.  
074 \* </p>  
075 \*  
076 \* @version $Id$  
077 \* @since 1.9  
078 \*/  
079public class FluentPropertyBeanIntrospector implements BeanIntrospector {  
080 /\*\* The default prefix for write methods. \*/  
081 public static final String DEFAULT\_WRITE\_METHOD\_PREFIX = "set";  
082  
083 /\*\* The logger. \*/  
084 private final Log log = LogFactory.getLog(getClass());  
085  
086 /\*\* The prefix of write methods to search for. \*/  
087 private final String writeMethodPrefix;  
088  
089 /\*\*  
090 \*  
091 \* Creates a new instance of <code>FluentPropertyBeanIntrospector</code> and  
092 \* initializes it with the prefix for write methods used by the classes to  
093 \* be inspected.  
094 \*  
095 \* @param writePrefix the prefix for write methods (must not be <b>null</b>)  
096 \* @throws IllegalArgumentException if the prefix is <b>null</b>  
097 \*/  
098 public FluentPropertyBeanIntrospector(final String writePrefix) {  
099 if (writePrefix == null) {  
100 throw new IllegalArgumentException(  
101 "Prefix for write methods must not be null!");  
102 }  
103 writeMethodPrefix = writePrefix;  
104 }  
105  
106 /\*\*  
107 \*  
108 \* Creates a new instance of <code>FluentPropertyBeanIntrospector</code> and  
109 \* sets the default prefix for write methods.  
110 \*/  
111 public FluentPropertyBeanIntrospector() {  
112 this(DEFAULT\_WRITE\_METHOD\_PREFIX);  
113 }  
114  
115 /\*\*  
116 \* Returns the prefix for write methods this instance scans for.  
117 \*  
118 \* @return the prefix for write methods  
119 \*/  
120 public String getWriteMethodPrefix() {  
121 return writeMethodPrefix;  
122 }  
123  
124 /\*\*  
125 \* Performs introspection. This method scans the current class's methods for  
126 \* property write methods which have not been discovered by default  
127 \* introspection.  
128 \*  
129 \* @param icontext the introspection context  
130 \* @throws IntrospectionException if an error occurs  
131 \*/  
132 public void introspect(final IntrospectionContext icontext)  
133 throws IntrospectionException {  
134 for (final Method m : icontext.getTargetClass().getMethods()) {  
135 if (m.getName().startsWith(getWriteMethodPrefix())) {  
136 final String propertyName = propertyName(m);  
137 final PropertyDescriptor pd = icontext  
138 .getPropertyDescriptor(propertyName);  
139 try {  
140 if (pd == null) {  
141 icontext.addPropertyDescriptor(createFluentPropertyDescritor(  
142 m, propertyName));  
143 } else if (pd.getWriteMethod() == null) {  
144 pd.setWriteMethod(m);  
145 }  
146 } catch (final IntrospectionException e) {  
147 log.info("Error when creating PropertyDescriptor for " + m  
148 + "! Ignoring this property.");  
149 log.debug("Exception is:", e);  
150 }  
151 }  
152 }  
153 }  
154  
155 /\*\*  
156 \* Derives the name of a property from the given set method.  
157 \*  
158 \* @param m the method  
159 \* @return the corresponding property name  
160 \*/  
161 private String propertyName(final Method m) {  
162 final String methodName = m.getName().substring(  
163 getWriteMethodPrefix().length());  
164 return (methodName.length() > 1) ? Introspector.decapitalize(methodName) : methodName  
165 .toLowerCase(Locale.ENGLISH);  
166 }  
167  
168 /\*\*  
169 \* Creates a property descriptor for a fluent API property.  
170 \*  
171 \* @param m the set method for the fluent API property  
172 \* @param propertyName the name of the corresponding property  
173 \* @return the descriptor  
174 \* @throws IntrospectionException if an error occurs  
175 \*/  
176 private PropertyDescriptor createFluentPropertyDescritor(final Method m,  
177 final String propertyName) throws IntrospectionException {  
178 return new PropertyDescriptor(propertyName(m), null, m);  
179 }  
180}