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 \*/  
017  
018package org.apache.commons.beanutils;  
019  
020import org.apache.commons.collections.Transformer;  
021import org.apache.commons.logging.Log;  
022import org.apache.commons.logging.LogFactory;  
023  
024import java.lang.reflect.InvocationTargetException;  
025  
026  
027/\*\*  
028 \* <p><code>Transformer</code> that outputs a property value.</p>  
029 \*  
030 \* <p>An implementation of <code>org.apache.commons.collections.Transformer</code> that transforms  
031 \* the object provided by returning the value of a specified property of the object. The  
032 \* constructor for <code>BeanToPropertyValueTransformer</code> requires the name of the property  
033 \* that will be used in the transformation. The property can be a simple, nested, indexed, or  
034 \* mapped property as defined by <code>org.apache.commons.beanutils.PropertyUtils</code>. If any  
035 \* object in the property path specified by <code>propertyName</code> is <code>null</code> then the  
036 \* outcome is based on the value of the <code>ignoreNull</code> attribute.  
037 \* </p>  
038 \*  
039 \* <p>  
040 \* A typical usage might look like:  
041 \* <code><pre>  
042 \* // create the transformer  
043 \* BeanToPropertyValueTransformer transformer = new BeanToPropertyValueTransformer( "person.address.city" );  
044 \*  
045 \* // transform the Collection  
046 \* Collection peoplesCities = CollectionUtils.collect( peopleCollection, transformer );  
047 \* </pre></code>  
048 \* </p>  
049 \*  
050 \* <p>  
051 \* This would take a <code>Collection</code> of person objects and return a <code>Collection</code>  
052 \* of objects which represents the cities in which each person lived. Assuming...  
053 \* <ul>  
054 \* <li>  
055 \* The top level object in the <code>peeopleCollection</code> is an object which represents a  
056 \* person.  
057 \* </li>  
058 \* <li>  
059 \* The person object has a <code>getAddress()</code> method which returns an object which  
060 \* represents a person's address.  
061 \* </li>  
062 \* <li>  
063 \* The address object has a <code>getCity()</code> method which returns an object which  
064 \* represents the city in which a person lives.  
065 \* </li>  
066 \* </ul>  
067 \*  
068 \* @version $Id$  
069 \* @see org.apache.commons.beanutils.PropertyUtils  
070 \* @see org.apache.commons.collections.Transformer  
071 \*/  
072public class BeanToPropertyValueTransformer implements Transformer {  
073  
074 /\*\* For logging. \*/  
075 private final Log log = LogFactory.getLog(this.getClass());  
076  
077 /\*\* The name of the property that will be used in the transformation of the object. \*/  
078 private String propertyName;  
079  
080 /\*\*  
081 \* <p>Should null objects on the property path throw an <code>IllegalArgumentException</code>?</p>  
082 \* <p>  
083 \* Determines whether <code>null</code> objects in the property path will genenerate an  
084 \* <code>IllegalArgumentException</code> or not. If set to <code>true</code> then if any objects  
085 \* in the property path evaluate to <code>null</code> then the  
086 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged but  
087 \* not rethrown and <code>null</code> will be returned. If set to <code>false</code> then if any  
088 \* objects in the property path evaluate to <code>null</code> then the  
089 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged and  
090 \* rethrown.  
091 \* </p>  
092 \*/  
093 private boolean ignoreNull;  
094  
095 /\*\*  
096 \* Constructs a Transformer which does not ignore nulls.  
097 \* Constructor which takes the name of the property that will be used in the transformation and  
098 \* assumes <code>ignoreNull</code> to be <code>false</code>.  
099 \*  
100 \* @param propertyName The name of the property that will be used in the transformation.  
101 \* @throws IllegalArgumentException If the <code>propertyName</code> is <code>null</code> or  
102 \* empty.  
103 \*/  
104 public BeanToPropertyValueTransformer(final String propertyName) {  
105 this(propertyName, false);  
106 }  
107  
108 /\*\*  
109 \* Constructs a Transformer and sets ignoreNull.  
110 \* Constructor which takes the name of the property that will be used in the transformation and  
111 \* a boolean which determines whether <code>null</code> objects in the property path will  
112 \* genenerate an <code>IllegalArgumentException</code> or not.  
113 \*  
114 \* @param propertyName The name of the property that will be used in the transformation.  
115 \* @param ignoreNull Determines whether <code>null</code> objects in the property path will  
116 \* genenerate an <code>IllegalArgumentException</code> or not.  
117 \* @throws IllegalArgumentException If the <code>propertyName</code> is <code>null</code> or  
118 \* empty.  
119 \*/  
120 public BeanToPropertyValueTransformer(final String propertyName, final boolean ignoreNull) {  
121 super();  
122  
123 if ((propertyName != null) && (propertyName.length() > 0)) {  
124 this.propertyName = propertyName;  
125 this.ignoreNull = ignoreNull;  
126 } else {  
127 throw new IllegalArgumentException(  
128 "propertyName cannot be null or empty");  
129 }  
130 }  
131  
132 /\*\*  
133 \* Returns the value of the property named in the transformer's constructor for  
134 \* the object provided. If any object in the property path leading up to the target property is  
135 \* <code>null</code> then the outcome will be based on the value of the <code>ignoreNull</code>  
136 \* attribute. By default, <code>ignoreNull</code> is <code>false</code> and would result in an  
137 \* <code>IllegalArgumentException</code> if an object in the property path leading up to the  
138 \* target property is <code>null</code>.  
139 \*  
140 \* @param object The object to be transformed.  
141 \* @return The value of the property named in the transformer's constructor for the object  
142 \* provided.  
143 \* @throws IllegalArgumentException If an IllegalAccessException, InvocationTargetException, or  
144 \* NoSuchMethodException is thrown when trying to access the property specified on the object  
145 \* provided. Or if an object in the property path provided is <code>null</code> and  
146 \* <code>ignoreNull</code> is set to <code>false</code>.  
147 \*/  
148 public Object transform(final Object object) {  
149  
150 Object propertyValue = null;  
151  
152 try {  
153 propertyValue = PropertyUtils.getProperty(object, propertyName);  
154 } catch (final IllegalArgumentException e) {  
155 final String errorMsg = "Problem during transformation. Null value encountered in property path...";  
156  
157 if (ignoreNull) {  
158 log.warn("WARNING: " + errorMsg + e);  
159 } else {  
160 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
161 if (!BeanUtils.initCause(iae, e)) {  
162 log.error(errorMsg, e);  
163 }  
164 throw iae;  
165 }  
166 } catch (final IllegalAccessException e) {  
167 final String errorMsg = "Unable to access the property provided.";  
168 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
169 if (!BeanUtils.initCause(iae, e)) {  
170 log.error(errorMsg, e);  
171 }  
172 throw iae;  
173 } catch (final InvocationTargetException e) {  
174 final String errorMsg = "Exception occurred in property's getter";  
175 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
176 if (!BeanUtils.initCause(iae, e)) {  
177 log.error(errorMsg, e);  
178 }  
179 throw iae;  
180 } catch (final NoSuchMethodException e) {  
181 final String errorMsg = "No property found for name [" +  
182 propertyName + "]";  
183 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
184 if (!BeanUtils.initCause(iae, e)) {  
185 log.error(errorMsg, e);  
186 }  
187 throw iae;  
188 }  
189  
190 return propertyValue;  
191 }  
192  
193 /\*\*  
194 \* Returns the name of the property that will be used in the transformation of the bean.  
195 \*  
196 \* @return The name of the property that will be used in the transformation of the bean.  
197 \*/  
198 public String getPropertyName() {  
199 return propertyName;  
200 }  
201  
202 /\*\*  
203 \* Returns the flag which determines whether <code>null</code> objects in the property path will  
204 \* genenerate an <code>IllegalArgumentException</code> or not. If set to <code>true</code> then  
205 \* if any objects in the property path evaluate to <code>null</code> then the  
206 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged but  
207 \* not rethrown and <code>null</code> will be returned. If set to <code>false</code> then if any  
208 \* objects in the property path evaluate to <code>null</code> then the  
209 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged and  
210 \* rethrown.  
211 \*  
212 \* @return The flag which determines whether <code>null</code> objects in the property path will  
213 \* genenerate an <code>IllegalArgumentException</code> or not.  
214 \*/  
215 public boolean isIgnoreNull() {  
216 return ignoreNull;  
217 }  
218}