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.Closure;  
021import org.apache.commons.logging.Log;  
022import org.apache.commons.logging.LogFactory;  
023  
024import java.lang.reflect.InvocationTargetException;  
025  
026  
027/\*\*  
028 \* <p><code>Closure</code> that sets a property.</p>  
029 \* <p>  
030 \* An implementation of <code>org.apache.commons.collections.Closure</code> that updates  
031 \* a specified property on the object provided with a specified value.  
032 \* The <code>BeanPropertyValueChangeClosure</code> constructor takes two parameters which determine  
033 \* what property will be updated and with what value.  
034 \* <dl>  
035 \* <dt>  
036 \* <b><code>  
037 \* <pre>public BeanPropertyValueChangeClosure( String propertyName, Object propertyValue )</pre>  
038 \* </code></b>  
039 \* </dt>  
040 \* <dd>  
041 \* Will create a <code>Closure</code> that will update an object by setting the property  
042 \* specified by <code>propertyName</code> to the value specified by <code>propertyValue</code>.  
043 \* </dd>  
044 \* </dl>  
045 \*  
046 \* <p/>  
047 \* <strong>Note:</strong> Property names can be a simple, nested, indexed, or mapped property as defined by  
048 \* <code>org.apache.commons.beanutils.PropertyUtils</code>. If any object in the property path  
049 \* specified by <code>propertyName</code> is <code>null</code> then the outcome is based on the  
050 \* value of the <code>ignoreNull</code> attribute.  
051 \*  
052 \* <p/>  
053 \* A typical usage might look like:  
054 \* <code><pre>  
055 \* // create the closure  
056 \* BeanPropertyValueChangeClosure closure =  
057 \* new BeanPropertyValueChangeClosure( "activeEmployee", Boolean.TRUE );  
058 \*  
059 \* // update the Collection  
060 \* CollectionUtils.forAllDo( peopleCollection, closure );  
061 \* </pre></code>  
062 \* <p/>  
063 \*  
064 \* This would take a <code>Collection</code> of person objects and update the  
065 \* <code>activeEmployee</code> property of each object in the <code>Collection</code> to  
066 \* <code>true</code>. Assuming...  
067 \* <ul>  
068 \* <li>  
069 \* The top level object in the <code>peopleCollection</code> is an object which represents a  
070 \* person.  
071 \* </li>  
072 \* <li>  
073 \* The person object has a <code>setActiveEmployee( boolean )</code> method which updates  
074 \* the value for the object's <code>activeEmployee</code> property.  
075 \* </li>  
076 \* </ul>  
077 \*  
078 \* @version $Id$  
079 \* @see org.apache.commons.beanutils.PropertyUtils  
080 \* @see org.apache.commons.collections.Closure  
081 \*/  
082public class BeanPropertyValueChangeClosure implements Closure {  
083  
084 /\*\* For logging. \*/  
085 private final Log log = LogFactory.getLog(this.getClass());  
086  
087 /\*\*  
088 \* The name of the property which will be updated when this <code>Closure</code> executes.  
089 \*/  
090 private String propertyName;  
091  
092 /\*\*  
093 \* The value that the property specified by <code>propertyName</code>  
094 \* will be updated to when this <code>Closure</code> executes.  
095 \*/  
096 private Object propertyValue;  
097  
098 /\*\*  
099 \* Determines whether <code>null</code> objects in the property path will genenerate an  
100 \* <code>IllegalArgumentException</code> or not. If set to <code>true</code> then if any objects  
101 \* in the property path leading up to the target property evaluate to <code>null</code> then the  
102 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged but  
103 \* not rethrown. If set to <code>false</code> then if any objects in the property path leading  
104 \* up to the target property evaluate to <code>null</code> then the  
105 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged and  
106 \* rethrown.  
107 \*/  
108 private boolean ignoreNull;  
109  
110 /\*\*  
111 \* Constructor which takes the name of the property to be changed, the new value to set  
112 \* the property to, and assumes <code>ignoreNull</code> to be <code>false</code>.  
113 \*  
114 \* @param propertyName The name of the property that will be updated with the value specified by  
115 \* <code>propertyValue</code>.  
116 \* @param propertyValue The value that <code>propertyName</code> will be set to on the target  
117 \* object.  
118 \* @throws IllegalArgumentException If the propertyName provided is null or empty.  
119 \*/  
120 public BeanPropertyValueChangeClosure(final String propertyName, final Object propertyValue) {  
121 this(propertyName, propertyValue, false);  
122 }  
123  
124 /\*\*  
125 \* Constructor which takes the name of the property to be changed, the new value to set  
126 \* the property to and a boolean which determines whether <code>null</code> objects in the  
127 \* property path will genenerate an <code>IllegalArgumentException</code> or not.  
128 \*  
129 \* @param propertyName The name of the property that will be updated with the value specified by  
130 \* <code>propertyValue</code>.  
131 \* @param propertyValue The value that <code>propertyName</code> will be set to on the target  
132 \* object.  
133 \* @param ignoreNull Determines whether <code>null</code> objects in the property path will  
134 \* genenerate an <code>IllegalArgumentException</code> or not.  
135 \* @throws IllegalArgumentException If the propertyName provided is null or empty.  
136 \*/  
137 public BeanPropertyValueChangeClosure(final String propertyName, final Object propertyValue, final boolean ignoreNull) {  
138 super();  
139  
140 if ((propertyName != null) && (propertyName.length() > 0)) {  
141 this.propertyName = propertyName;  
142 this.propertyValue = propertyValue;  
143 this.ignoreNull = ignoreNull;  
144 } else {  
145 throw new IllegalArgumentException("propertyName cannot be null or empty");  
146 }  
147 }  
148  
149 /\*\*  
150 \* Updates the target object provided using the property update criteria provided when this  
151 \* <code>BeanPropertyValueChangeClosure</code> was constructed. If any object in the property  
152 \* path leading up to the target property is <code>null</code> then the outcome will be based on  
153 \* the value of the <code>ignoreNull</code> attribute. By default, <code>ignoreNull</code> is  
154 \* <code>false</code> and would result in an <code>IllegalArgumentException</code> if an object  
155 \* in the property path leading up to the target property is <code>null</code>.  
156 \*  
157 \* @param object The object to be updated.  
158 \* @throws IllegalArgumentException If an IllegalAccessException, InvocationTargetException, or  
159 \* NoSuchMethodException is thrown when trying to access the property specified on the object  
160 \* provided. Or if an object in the property path provided is <code>null</code> and  
161 \* <code>ignoreNull</code> is set to <code>false</code>.  
162 \*/  
163 public void execute(final Object object) {  
164  
165 try {  
166 PropertyUtils.setProperty(object, propertyName, propertyValue);  
167 } catch (final IllegalArgumentException e) {  
168 final String errorMsg = "Unable to execute Closure. Null value encountered in property path...";  
169  
170 if (ignoreNull) {  
171 log.warn("WARNING: " + errorMsg + e);  
172 } else {  
173 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
174 if (!BeanUtils.initCause(iae, e)) {  
175 log.error(errorMsg, e);  
176 }  
177 throw iae;  
178 }  
179 } catch (final IllegalAccessException e) {  
180 final String errorMsg = "Unable to access the property provided.";  
181 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
182 if (!BeanUtils.initCause(iae, e)) {  
183 log.error(errorMsg, e);  
184 }  
185 throw iae;  
186 } catch (final InvocationTargetException e) {  
187 final String errorMsg = "Exception occurred in property's getter";  
188 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
189 if (!BeanUtils.initCause(iae, e)) {  
190 log.error(errorMsg, e);  
191 }  
192 throw iae;  
193 } catch (final NoSuchMethodException e) {  
194 final String errorMsg = "Property not found";  
195 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
196 if (!BeanUtils.initCause(iae, e)) {  
197 log.error(errorMsg, e);  
198 }  
199 throw iae;  
200 }  
201 }  
202  
203 /\*\*  
204 \* Returns the name of the property which will be updated when this <code>Closure</code> executes.  
205 \*  
206 \* @return The name of the property which will be updated when this <code>Closure</code> executes.  
207 \*/  
208 public String getPropertyName() {  
209 return propertyName;  
210 }  
211  
212 /\*\*  
213 \* Returns the value that the property specified by <code>propertyName</code>  
214 \* will be updated to when this <code>Closure</code> executes.  
215 \*  
216 \* @return The value that the property specified by <code>propertyName</code>  
217 \* will be updated to when this <code>Closure</code> executes.  
218 \*/  
219 public Object getPropertyValue() {  
220 return propertyValue;  
221 }  
222  
223 /\*\*  
224 \* Returns the flag that determines whether <code>null</code> objects in the property path will  
225 \* genenerate an <code>IllegalArgumentException</code> or not. If set to <code>true</code> then  
226 \* if any objects in the property path leading up to the target property evaluate to  
227 \* <code>null</code> then the <code>IllegalArgumentException</code> throw by  
228 \* <code>PropertyUtils</code> will be logged but not rethrown. If set to <code>false</code> then  
229 \* if any objects in the property path leading up to the target property evaluate to  
230 \* <code>null</code> then the <code>IllegalArgumentException</code> throw by  
231 \* <code>PropertyUtils</code> will be logged and rethrown.  
232 \*  
233 \* @return The flag that determines whether <code>null</code> objects in the property path will  
234 \* genenerate an <code>IllegalArgumentException</code> or not.  
235 \*/  
236 public boolean isIgnoreNull() {  
237 return ignoreNull;  
238 }  
239}