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 java.lang.reflect.InvocationTargetException;  
021  
022import org.apache.commons.collections.Predicate;  
023import org.apache.commons.logging.Log;  
024import org.apache.commons.logging.LogFactory;  
025  
026  
027/\*\*  
028 \* <p><code>Predicate</code> that evaluates a property value against a specified value.</p>  
029 \* <p>  
030 \* An implementation of <code>org.apache.commons.collections.Predicate</code> that evaluates a  
031 \* property value on the object provided against a specified value and returns <code>true</code>  
032 \* if equal; <code>false</code> otherwise.  
033 \* The <code>BeanPropertyValueEqualsPredicate</code> constructor takes two parameters which  
034 \* determine what property will be evaluated on the target object and what its expected value should  
035 \* be.  
036 \* <dl>  
037 \* <dt>  
038 \* <strong><code>  
039 \* <pre>public BeanPropertyValueEqualsPredicate( String propertyName, Object propertyValue )</pre>  
040 \* </code></strong>  
041 \* </dt>  
042 \* <dd>  
043 \* Will create a <code>Predicate</code> that will evaluate the target object and return  
044 \* <code>true</code> if the property specified by <code>propertyName</code> has a value which  
045 \* is equal to the the value specified by <code>propertyValue</code>. Or return  
046 \* <code>false</code> otherwise.  
047 \* </dd>  
048 \* </dl>  
049 \* </p>  
050 \* <p>  
051 \* <strong>Note:</strong> Property names can be a simple, nested, indexed, or mapped property as defined by  
052 \* <code>org.apache.commons.beanutils.PropertyUtils</code>. If any object in the property path  
053 \* specified by <code>propertyName</code> is <code>null</code> then the outcome is based on the  
054 \* value of the <code>ignoreNull</code> attribute.  
055 \* </p>  
056 \* <p>  
057 \* A typical usage might look like:  
058 \* <code><pre>  
059 \* // create the closure  
060 \* BeanPropertyValueEqualsPredicate predicate =  
061 \* new BeanPropertyValueEqualsPredicate( "activeEmployee", Boolean.FALSE );  
062 \*  
063 \* // filter the Collection  
064 \* CollectionUtils.filter( peopleCollection, predicate );  
065 \* </pre></code>  
066 \* </p>  
067 \* <p>  
068 \* This would take a <code>Collection</code> of person objects and filter out any people whose  
069 \* <code>activeEmployee</code> property is <code>false</code>. Assuming...  
070 \* <ul>  
071 \* <li>  
072 \* The top level object in the <code>peeopleCollection</code> is an object which represents a  
073 \* person.  
074 \* </li>  
075 \* <li>  
076 \* The person object has a <code>getActiveEmployee()</code> method which returns  
077 \* the boolean value for the object's <code>activeEmployee</code> property.  
078 \* </li>  
079 \* </ul>  
080 \* </p>  
081 \* <p>  
082 \* Another typical usage might look like:  
083 \* <code><pre>  
084 \* // create the closure  
085 \* BeanPropertyValueEqualsPredicate predicate =  
086 \* new BeanPropertyValueEqualsPredicate( "personId", "456-12-1234" );  
087 \*  
088 \* // search the Collection  
089 \* CollectionUtils.find( peopleCollection, predicate );  
090 \* </pre></code>  
091 \* </p>  
092 \* <p>  
093 \* This would search a <code>Collection</code> of person objects and return the first object whose  
094 \* <code>personId</code> property value equals <code>456-12-1234</code>. Assuming...  
095 \* <ul>  
096 \* <li>  
097 \* The top level object in the <code>peeopleCollection</code> is an object which represents a  
098 \* person.  
099 \* </li>  
100 \* <li>  
101 \* The person object has a <code>getPersonId()</code> method which returns  
102 \* the value for the object's <code>personId</code> property.  
103 \* </li>  
104 \* </ul>  
105 \* </p>  
106 \*  
107 \* @version $Id$  
108 \* @see org.apache.commons.beanutils.PropertyUtils  
109 \* @see org.apache.commons.collections.Predicate  
110 \*/  
111public class BeanPropertyValueEqualsPredicate implements Predicate {  
112  
113 /\*\* For logging. \*/  
114 private final Log log = LogFactory.getLog(this.getClass());  
115  
116 /\*\*  
117 \* The name of the property which will be evaluated when this <code>Predicate</code> is executed.  
118 \*/  
119 private String propertyName;  
120  
121 /\*\*  
122 \* The value that the property specified by <code>propertyName</code>  
123 \* will be compared to when this <code>Predicate</code> executes.  
124 \*/  
125 private Object propertyValue;  
126  
127 /\*\*  
128 \* <p>Should <code>null</code> objects in the property path be ignored?</p>  
129 \* <p>  
130 \* Determines whether <code>null</code> objects in the property path will genenerate an  
131 \* <code>IllegalArgumentException</code> or not. If set to <code>true</code> then if any objects  
132 \* in the property path evaluate to <code>null</code> then the  
133 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged but  
134 \* not rethrown and <code>false</code> will be returned. If set to <code>false</code> then if  
135 \* any objects in the property path evaluate to <code>null</code> then the  
136 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged and  
137 \* rethrown.  
138 \* </p>  
139 \*/  
140 private boolean ignoreNull;  
141  
142 /\*\*  
143 \* Constructor which takes the name of the property, its expected value to be used in evaluation,  
144 \* and assumes <code>ignoreNull</code> to be <code>false</code>.  
145 \*  
146 \* @param propertyName The name of the property that will be evaluated against the expected value.  
147 \* @param propertyValue The value to use in object evaluation.  
148 \* @throws IllegalArgumentException If the property name provided is null or empty.  
149 \*/  
150 public BeanPropertyValueEqualsPredicate(final String propertyName, final Object propertyValue) {  
151 this(propertyName, propertyValue, false);  
152 }  
153  
154 /\*\*  
155 \* Constructor which takes the name of the property, its expected value  
156 \* to be used in evaluation, and a boolean which determines whether <code>null</code> objects in  
157 \* the property path will genenerate an <code>IllegalArgumentException</code> or not.  
158 \*  
159 \* @param propertyName The name of the property that will be evaluated against the expected value.  
160 \* @param propertyValue The value to use in object evaluation.  
161 \* @param ignoreNull Determines whether <code>null</code> objects in the property path will  
162 \* genenerate an <code>IllegalArgumentException</code> or not.  
163 \* @throws IllegalArgumentException If the property name provided is null or empty.  
164 \*/  
165 public BeanPropertyValueEqualsPredicate(final String propertyName, final Object propertyValue, final boolean ignoreNull) {  
166 super();  
167  
168 if ((propertyName != null) && (propertyName.length() > 0)) {  
169 this.propertyName = propertyName;  
170 this.propertyValue = propertyValue;  
171 this.ignoreNull = ignoreNull;  
172 } else {  
173 throw new IllegalArgumentException("propertyName cannot be null or empty");  
174 }  
175 }  
176  
177 /\*\*  
178 \* Evaulates the object provided against the criteria specified when this  
179 \* <code>BeanPropertyValueEqualsPredicate</code> was constructed. Equality is based on  
180 \* either reference or logical equality as defined by the property object's equals method. If  
181 \* any object in the property path leading up to the target property is <code>null</code> then  
182 \* the outcome will be based on the value of the <code>ignoreNull</code> attribute. By default,  
183 \* <code>ignoreNull</code> is <code>false</code> and would result in an  
184 \* <code>IllegalArgumentException</code> if an object in the property path leading up to the  
185 \* target property is <code>null</code>.  
186 \*  
187 \* @param object The object to be evaluated.  
188 \* @return True if the object provided meets all the criteria for this <code>Predicate</code>;  
189 \* false otherwise.  
190 \* @throws IllegalArgumentException If an IllegalAccessException, InvocationTargetException, or  
191 \* NoSuchMethodException is thrown when trying to access the property specified on the object  
192 \* provided. Or if an object in the property path provided is <code>null</code> and  
193 \* <code>ignoreNull</code> is set to <code>false</code>.  
194 \*/  
195 public boolean evaluate(final Object object) {  
196  
197 boolean evaluation = false;  
198  
199 try {  
200 evaluation = evaluateValue(propertyValue,  
201 PropertyUtils.getProperty(object, propertyName));  
202 } catch (final IllegalArgumentException e) {  
203 final String errorMsg = "Problem during evaluation. Null value encountered in property path...";  
204  
205 if (ignoreNull) {  
206 log.warn("WARNING: " + errorMsg + e);  
207 } else {  
208 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
209 if (!BeanUtils.initCause(iae, e)) {  
210 log.error(errorMsg, e);  
211 }  
212 throw iae;  
213 }  
214 } catch (final IllegalAccessException e) {  
215 final String errorMsg = "Unable to access the property provided.";  
216 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
217 if (!BeanUtils.initCause(iae, e)) {  
218 log.error(errorMsg, e);  
219 }  
220 throw iae;  
221 } catch (final InvocationTargetException e) {  
222 final String errorMsg = "Exception occurred in property's getter";  
223 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
224 if (!BeanUtils.initCause(iae, e)) {  
225 log.error(errorMsg, e);  
226 }  
227 throw iae;  
228 } catch (final NoSuchMethodException e) {  
229 final String errorMsg = "Property not found.";  
230 final IllegalArgumentException iae = new IllegalArgumentException(errorMsg);  
231 if (!BeanUtils.initCause(iae, e)) {  
232 log.error(errorMsg, e);  
233 }  
234 throw iae;  
235 }  
236  
237 return evaluation;  
238 }  
239  
240 /\*\*  
241 \* Utility method which evaluates whether the actual property value equals the expected property  
242 \* value.  
243 \*  
244 \* @param expected The expected value.  
245 \* @param actual The actual value.  
246 \* @return True if they are equal; false otherwise.  
247 \*/  
248 protected boolean evaluateValue(final Object expected, final Object actual) {  
249 return (expected == actual) || ((expected != null) && expected.equals(actual));  
250 }  
251  
252 /\*\*  
253 \* Returns the name of the property which will be evaluated when this <code>Predicate</code> is  
254 \* executed.  
255 \*  
256 \* @return The name of the property which will be evaluated when this <code>Predicate</code> is  
257 \* executed.  
258 \*/  
259 public String getPropertyName() {  
260 return propertyName;  
261 }  
262  
263 /\*\*  
264 \* Returns the value that the property specified by <code>propertyName</code> will be compared to  
265 \* when this <code>Predicate</code> executes.  
266 \*  
267 \* @return The value that the property specified by <code>propertyName</code> will be compared to  
268 \* when this <code>Predicate</code> executes.  
269 \*/  
270 public Object getPropertyValue() {  
271 return propertyValue;  
272 }  
273  
274 /\*\*  
275 \* Returns the flag which determines whether <code>null</code> objects in the property path will  
276 \* genenerate an <code>IllegalArgumentException</code> or not. If set to <code>true</code> then  
277 \* if any objects in the property path evaluate to <code>null</code> then the  
278 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged but  
279 \* not rethrown and <code>false</code> will be returned. If set to <code>false</code> then if  
280 \* any objects in the property path evaluate to <code>null</code> then the  
281 \* <code>IllegalArgumentException</code> throw by <code>PropertyUtils</code> will be logged and  
282 \* rethrown.  
283 \*  
284 \* @return The flag which determines whether <code>null</code> objects in the property path will  
285 \* genenerate an <code>IllegalArgumentException</code> or not.  
286 \*/  
287 public boolean isIgnoreNull() {  
288 return ignoreNull;  
289 }  
290}