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  
018  
019package org.apache.commons.beanutils;  
020  
021import java.io.Serializable;  
022import java.lang.reflect.InvocationTargetException;  
023  
024  
025/\*\*  
026 \* <p>Implementation of <code>DynaBean</code> that wraps a standard JavaBean  
027 \* instance, so that DynaBean APIs can be used to access its properties.</p>  
028 \*  
029 \* <p>  
030 \* The most common use cases for this class involve wrapping an existing java bean.  
031 \* (This makes it different from the typical use cases for other <code>DynaBean</code>'s.)  
032 \* For example:  
033 \* </p>  
034 \* <code><pre>  
035 \* Object aJavaBean = ...;  
036 \* ...  
037 \* DynaBean db = new WrapDynaBean(aJavaBean);  
038 \* ...  
039 \* </pre></code>  
040 \*  
041 \* <p><strong>IMPLEMENTATION NOTE</strong> - This implementation does not  
042 \* support the <code>contains()</code> and <code>remove()</code> methods.</p>  
043 \*  
044 \* @version $Id$  
045 \*/  
046  
047public class WrapDynaBean implements DynaBean, Serializable {  
048  
049  
050 // ---------------------------------------------------------- Constructors  
051  
052  
053 /\*\*  
054 \* Construct a new <code>DynaBean</code> associated with the specified  
055 \* JavaBean instance.  
056 \*  
057 \* @param instance JavaBean instance to be wrapped  
058 \*/  
059 public WrapDynaBean(final Object instance) {  
060  
061 this(instance, null);  
062  
063 }  
064  
065 /\*\*  
066 \* Creates a new instance of {@code WrapDynaBean}, associates it with the specified  
067 \* JavaBean instance, and initializes the bean's {@code DynaClass}. Using this  
068 \* constructor this {@code WrapDynaBean} instance can be assigned a class which has  
069 \* been configured externally. If no {@code WrapDynaClass} is provided, a new one is  
070 \* created using a standard mechanism.  
071 \*  
072 \* @param instance JavaBean instance to be wrapped  
073 \* @param cls the optional {@code WrapDynaClass} to be used for this bean  
074 \* @since 1.9  
075 \*/  
076 public WrapDynaBean(final Object instance, final WrapDynaClass cls) {  
077  
078 this.instance = instance;  
079 this.dynaClass = (cls != null) ? cls : (WrapDynaClass) getDynaClass();  
080  
081 }  
082  
083 // ---------------------------------------------------- Instance Variables  
084  
085  
086 /\*\*  
087 \* The <code>DynaClass</code> "base class" that this DynaBean  
088 \* is associated with.  
089 \*/  
090 protected transient WrapDynaClass dynaClass = null;  
091  
092  
093 /\*\*  
094 \* The JavaBean instance wrapped by this WrapDynaBean.  
095 \*/  
096 protected Object instance = null;  
097  
098  
099 // ------------------------------------------------------ DynaBean Methods  
100  
101  
102 /\*\*  
103 \* Does the specified mapped property contain a value for the specified  
104 \* key value?  
105 \*  
106 \* @param name Name of the property to check  
107 \* @param key Name of the key to check  
108 \* @return <code>true<code> if the mapped property contains a value for  
109 \* the specified key, otherwise <code>false</code>  
110 \*  
111 \* @throws IllegalArgumentException if there is no property  
112 \* of the specified name  
113 \*/  
114 public boolean contains(final String name, final String key) {  
115  
116 throw new UnsupportedOperationException  
117 ("WrapDynaBean does not support contains()");  
118  
119 }  
120  
121  
122 /\*\*  
123 \* Return the value of a simple property with the specified name.  
124 \*  
125 \* @param name Name of the property whose value is to be retrieved  
126 \* @return The property's value  
127 \*  
128 \* @throws IllegalArgumentException if there is no property  
129 \* of the specified name  
130 \*/  
131 public Object get(final String name) {  
132  
133 Object value = null;  
134 try {  
135 value = getPropertyUtils().getSimpleProperty(instance, name);  
136 } catch (final InvocationTargetException ite) {  
137 final Throwable cause = ite.getTargetException();  
138 throw new IllegalArgumentException  
139 ("Error reading property '" + name +  
140 "' nested exception - " + cause);  
141 } catch (final Throwable t) {  
142 throw new IllegalArgumentException  
143 ("Error reading property '" + name +  
144 "', exception - " + t);  
145 }  
146 return (value);  
147  
148 }  
149  
150  
151 /\*\*  
152 \* Return the value of an indexed property with the specified name.  
153 \*  
154 \* @param name Name of the property whose value is to be retrieved  
155 \* @param index Index of the value to be retrieved  
156 \* @return The indexed property's value  
157 \*  
158 \* @throws IllegalArgumentException if there is no property  
159 \* of the specified name  
160 \* @throws IllegalArgumentException if the specified property  
161 \* exists, but is not indexed  
162 \* @throws IndexOutOfBoundsException if the specified index  
163 \* is outside the range of the underlying property  
164 \* @throws NullPointerException if no array or List has been  
165 \* initialized for this property  
166 \*/  
167 public Object get(final String name, final int index) {  
168  
169 Object value = null;  
170 try {  
171 value = getPropertyUtils().getIndexedProperty(instance, name, index);  
172 } catch (final IndexOutOfBoundsException e) {  
173 throw e;  
174 } catch (final InvocationTargetException ite) {  
175 final Throwable cause = ite.getTargetException();  
176 throw new IllegalArgumentException  
177 ("Error reading indexed property '" + name +  
178 "' nested exception - " + cause);  
179 } catch (final Throwable t) {  
180 throw new IllegalArgumentException  
181 ("Error reading indexed property '" + name +  
182 "', exception - " + t);  
183 }  
184 return (value);  
185  
186 }  
187  
188  
189 /\*\*  
190 \* Return the value of a mapped property with the specified name,  
191 \* or <code>null</code> if there is no value for the specified key.  
192 \*  
193 \* @param name Name of the property whose value is to be retrieved  
194 \* @param key Key of the value to be retrieved  
195 \* @return The mapped property's value  
196 \*  
197 \* @throws IllegalArgumentException if there is no property  
198 \* of the specified name  
199 \* @throws IllegalArgumentException if the specified property  
200 \* exists, but is not mapped  
201 \*/  
202 public Object get(final String name, final String key) {  
203  
204 Object value = null;  
205 try {  
206 value = getPropertyUtils().getMappedProperty(instance, name, key);  
207 } catch (final InvocationTargetException ite) {  
208 final Throwable cause = ite.getTargetException();  
209 throw new IllegalArgumentException  
210 ("Error reading mapped property '" + name +  
211 "' nested exception - " + cause);  
212 } catch (final Throwable t) {  
213 throw new IllegalArgumentException  
214 ("Error reading mapped property '" + name +  
215 "', exception - " + t);  
216 }  
217 return (value);  
218  
219 }  
220  
221  
222 /\*\*  
223 \* Return the <code>DynaClass</code> instance that describes the set of  
224 \* properties available for this DynaBean.  
225 \* @return The associated DynaClass  
226 \*/  
227 public DynaClass getDynaClass() {  
228  
229 if (dynaClass == null) {  
230 dynaClass = WrapDynaClass.createDynaClass(instance.getClass());  
231 }  
232  
233 return (this.dynaClass);  
234  
235 }  
236  
237  
238 /\*\*  
239 \* Remove any existing value for the specified key on the  
240 \* specified mapped property.  
241 \*  
242 \* @param name Name of the property for which a value is to  
243 \* be removed  
244 \* @param key Key of the value to be removed  
245 \*  
246 \* @throws IllegalArgumentException if there is no property  
247 \* of the specified name  
248 \*/  
249 public void remove(final String name, final String key) {  
250  
251  
252 throw new UnsupportedOperationException  
253 ("WrapDynaBean does not support remove()");  
254  
255 }  
256  
257  
258 /\*\*  
259 \* Set the value of a simple property with the specified name.  
260 \*  
261 \* @param name Name of the property whose value is to be set  
262 \* @param value Value to which this property is to be set  
263 \*  
264 \* @throws ConversionException if the specified value cannot be  
265 \* converted to the type required for this property  
266 \* @throws IllegalArgumentException if there is no property  
267 \* of the specified name  
268 \* @throws NullPointerException if an attempt is made to set a  
269 \* primitive property to null  
270 \*/  
271 public void set(final String name, final Object value) {  
272  
273 try {  
274 getPropertyUtils().setSimpleProperty(instance, name, value);  
275 } catch (final InvocationTargetException ite) {  
276 final Throwable cause = ite.getTargetException();  
277 throw new IllegalArgumentException  
278 ("Error setting property '" + name +  
279 "' nested exception -" + cause);  
280 } catch (final Throwable t) {  
281 throw new IllegalArgumentException  
282 ("Error setting property '" + name +  
283 "', exception - " + t);  
284 }  
285  
286 }  
287  
288  
289 /\*\*  
290 \* Set the value of an indexed property with the specified name.  
291 \*  
292 \* @param name Name of the property whose value is to be set  
293 \* @param index Index of the property to be set  
294 \* @param value Value to which this property is to be set  
295 \*  
296 \* @throws ConversionException if the specified value cannot be  
297 \* converted to the type required for this property  
298 \* @throws IllegalArgumentException if there is no property  
299 \* of the specified name  
300 \* @throws IllegalArgumentException if the specified property  
301 \* exists, but is not indexed  
302 \* @throws IndexOutOfBoundsException if the specified index  
303 \* is outside the range of the underlying property  
304 \*/  
305 public void set(final String name, final int index, final Object value) {  
306  
307 try {  
308 getPropertyUtils().setIndexedProperty(instance, name, index, value);  
309 } catch (final IndexOutOfBoundsException e) {  
310 throw e;  
311 } catch (final InvocationTargetException ite) {  
312 final Throwable cause = ite.getTargetException();  
313 throw new IllegalArgumentException  
314 ("Error setting indexed property '" + name +  
315 "' nested exception - " + cause);  
316 } catch (final Throwable t) {  
317 throw new IllegalArgumentException  
318 ("Error setting indexed property '" + name +  
319 "', exception - " + t);  
320 }  
321  
322 }  
323  
324  
325 /\*\*  
326 \* Set the value of a mapped property with the specified name.  
327 \*  
328 \* @param name Name of the property whose value is to be set  
329 \* @param key Key of the property to be set  
330 \* @param value Value to which this property is to be set  
331 \*  
332 \* @throws ConversionException if the specified value cannot be  
333 \* converted to the type required for this property  
334 \* @throws IllegalArgumentException if there is no property  
335 \* of the specified name  
336 \* @throws IllegalArgumentException if the specified property  
337 \* exists, but is not mapped  
338 \*/  
339 public void set(final String name, final String key, final Object value) {  
340  
341 try {  
342 getPropertyUtils().setMappedProperty(instance, name, key, value);  
343 } catch (final InvocationTargetException ite) {  
344 final Throwable cause = ite.getTargetException();  
345 throw new IllegalArgumentException  
346 ("Error setting mapped property '" + name +  
347 "' nested exception - " + cause);  
348 } catch (final Throwable t) {  
349 throw new IllegalArgumentException  
350 ("Error setting mapped property '" + name +  
351 "', exception - " + t);  
352 }  
353  
354 }  
355  
356 /\*\*  
357 \* Gets the bean instance wrapped by this DynaBean.  
358 \* For most common use cases,  
359 \* this object should already be known  
360 \* and this method safely be ignored.  
361 \* But some creators of frameworks using <code>DynaBean</code>'s may  
362 \* find this useful.  
363 \*  
364 \* @return the java bean Object wrapped by this <code>DynaBean</code>  
365 \*/  
366 public Object getInstance() {  
367 return instance;  
368 }  
369  
370  
371 // ------------------------------------------------------ Protected Methods  
372  
373  
374 /\*\*  
375 \* Return the property descriptor for the specified property name.  
376 \*  
377 \* @param name Name of the property for which to retrieve the descriptor  
378 \* @return The descriptor for the specified property  
379 \*  
380 \* @throws IllegalArgumentException if this is not a valid property  
381 \* name for our DynaClass  
382 \*/  
383 protected DynaProperty getDynaProperty(final String name) {  
384  
385 final DynaProperty descriptor = getDynaClass().getDynaProperty(name);  
386 if (descriptor == null) {  
387 throw new IllegalArgumentException  
388 ("Invalid property name '" + name + "'");  
389 }  
390 return (descriptor);  
391  
392 }  
393  
394 /\*\*  
395 \* Returns the {@code PropertyUtilsBean} instance to be used for accessing properties.  
396 \* If available, this object is obtained from the associated {@code WrapDynaClass}.  
397 \*  
398 \* @return the associated {@code PropertyUtilsBean}  
399 \*/  
400 private PropertyUtilsBean getPropertyUtils() {  
401  
402 PropertyUtilsBean propUtils = null;  
403 if (dynaClass != null) {  
404 propUtils = dynaClass.getPropertyUtilsBean();  
405 }  
406 return (propUtils != null) ? propUtils : PropertyUtilsBean.getInstance();  
407  
408 }  
409}