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  
019/\*\*  
020 \* <p>DynaClass which implements the <code>MutableDynaClass</code> interface.</p>  
021 \*  
022 \* <p>A <code>MutableDynaClass</code> is a specialized extension to <code>DynaClass</code>  
023 \* that allows properties to be added or removed dynamically.</p>  
024 \*  
025 \* <p>This implementation has one slightly unusual default behaviour - calling  
026 \* the <code>getDynaProperty(name)</code> method for a property which doesn't  
027 \* exist returns a <code>DynaProperty</code> rather than <code>null</code>. The  
028 \* reason for this is that <code>BeanUtils</code> calls this method to check if  
029 \* a property exists before trying to set the value. This would defeat the object  
030 \* of the <code>LazyDynaBean</code> which automatically adds missing properties  
031 \* when any of its <code>set()</code> methods are called. For this reason the  
032 \* <code>isDynaProperty(name)</code> method has been added to this implementation  
033 \* in order to determine if a property actually exists. If the more <i>normal</i>  
034 \* behaviour of returning <code>null</code> is required, then this can be achieved  
035 \* by calling the <code>setReturnNull(true)</code>.</p>  
036 \*  
037 \* <p>The <code>add(name, type, readable, writable)</code> method is not implemented  
038 \* and always throws an <code>UnsupportedOperationException</code>. I believe  
039 \* this attributes need to be added to the <code>DynaProperty</code> class  
040 \* in order to control read/write facilities.</p>  
041 \*  
042 \* @version $Id$  
043 \* @see LazyDynaBean  
044 \*/  
045public class LazyDynaClass extends BasicDynaClass implements MutableDynaClass {  
046  
047 /\*\*  
048 \* Controls whether changes to this DynaClass's properties are allowed.  
049 \*/  
050 protected boolean restricted;  
051  
052 /\*\*  
053 \* <p>Controls whether the <code>getDynaProperty()</code> method returns  
054 \* null if a property doesn't exist - or creates a new one.</p>  
055 \*  
056 \* <p>Default is <code>false</code>.  
057 \*/  
058 protected boolean returnNull = false;  
059  
060 /\*\*  
061 \* Construct a new LazyDynaClass with default parameters.  
062 \*/  
063 public LazyDynaClass() {  
064 this(null, (DynaProperty[])null);  
065 }  
066  
067 /\*\*  
068 \* Construct a new LazyDynaClass with the specified name.  
069 \*  
070 \* @param name Name of this DynaBean class  
071 \*/  
072 public LazyDynaClass(final String name) {  
073 this(name, (DynaProperty[])null);  
074 }  
075  
076 /\*\*  
077 \* Construct a new LazyDynaClass with the specified name and DynaBean class.  
078 \*  
079 \* @param name Name of this DynaBean class  
080 \* @param dynaBeanClass The implementation class for new instances  
081 \*/  
082 public LazyDynaClass(final String name, final Class<?> dynaBeanClass) {  
083 this(name, dynaBeanClass, null);  
084 }  
085  
086 /\*\*  
087 \* Construct a new LazyDynaClass with the specified name and properties.  
088 \*  
089 \* @param name Name of this DynaBean class  
090 \* @param properties Property descriptors for the supported properties  
091 \*/  
092 public LazyDynaClass(final String name, final DynaProperty[] properties) {  
093 this(name, LazyDynaBean.class, properties);  
094 }  
095  
096 /\*\*  
097 \* Construct a new LazyDynaClass with the specified name, DynaBean class and properties.  
098 \*  
099 \* @param name Name of this DynaBean class  
100 \* @param dynaBeanClass The implementation class for new instances  
101 \* @param properties Property descriptors for the supported properties  
102 \*/  
103 public LazyDynaClass(final String name, final Class<?> dynaBeanClass, final DynaProperty properties[]) {  
104 super(name, dynaBeanClass, properties);  
105 }  
106  
107 /\*\*  
108 \* <p>Is this DynaClass currently restricted.</p>  
109 \* <p>If restricted, no changes to the existing registration of  
110 \* property names, data types, readability, or writeability are allowed.</p>  
111 \* @return <code>true</code> if this {@link MutableDynaClass} cannot be changed  
112 \* otherwise <code>false</code>  
113 \*/  
114 public boolean isRestricted() {  
115 return restricted;  
116 }  
117  
118 /\*\*  
119 \* <p>Set whether this DynaClass is currently restricted.</p>  
120 \* <p>If restricted, no changes to the existing registration of  
121 \* property names, data types, readability, or writeability are allowed.</p>  
122 \* @param restricted <code>true</code> if this {@link MutableDynaClass} cannot  
123 \* be changed otherwise <code>false</code>  
124 \*/  
125 public void setRestricted(final boolean restricted) {  
126 this.restricted = restricted;  
127 }  
128  
129 /\*\*  
130 \* Should this DynaClass return a <code>null</code> from  
131 \* the <code>getDynaProperty(name)</code> method if the property  
132 \* doesn't exist.  
133 \*  
134 \* @return <code>true</code> if a <code>null</code> {@link DynaProperty}  
135 \* should be returned if the property doesn't exist, otherwise  
136 \* <code>false</code> if a new {@link DynaProperty} should be created.  
137 \*/  
138 public boolean isReturnNull() {  
139 return returnNull;  
140 }  
141  
142 /\*\*  
143 \* Set whether this DynaClass should return a <code>null</code> from  
144 \* the <code>getDynaProperty(name)</code> method if the property  
145 \* doesn't exist.  
146 \* @param returnNull <code>true</code> if a <code>null</code> {@link DynaProperty}  
147 \* should be returned if the property doesn't exist, otherwise  
148 \* <code>false</code> if a new {@link DynaProperty} should be created.  
149 \*/  
150 public void setReturnNull(final boolean returnNull) {  
151 this.returnNull = returnNull;  
152 }  
153  
154 /\*\*  
155 \* Add a new dynamic property with no restrictions on data type,  
156 \* readability, or writeability.  
157 \*  
158 \* @param name Name of the new dynamic property  
159 \*  
160 \* @throws IllegalArgumentException if name is null  
161 \* @throws IllegalStateException if this DynaClass is currently  
162 \* restricted, so no new properties can be added  
163 \*/  
164 public void add(final String name) {  
165 add(new DynaProperty(name));  
166 }  
167  
168 /\*\*  
169 \* Add a new dynamic property with the specified data type, but with  
170 \* no restrictions on readability or writeability.  
171 \*  
172 \* @param name Name of the new dynamic property  
173 \* @param type Data type of the new dynamic property (null for no  
174 \* restrictions)  
175 \*  
176 \* @throws IllegalArgumentException if name is null  
177 \* @throws IllegalStateException if this DynaClass is currently  
178 \* restricted, so no new properties can be added  
179 \*/  
180 public void add(final String name, final Class<?> type) {  
181 if (type == null) {  
182 add(name);  
183 } else {  
184 add(new DynaProperty(name, type));  
185 }  
186 }  
187  
188 /\*\*  
189 \* <p>Add a new dynamic property with the specified data type, readability,  
190 \* and writeability.</p>  
191 \*  
192 \* <p><strong>N.B.</strong>Support for readable/writeable properties has not been implemented  
193 \* and this method always throws a <code>UnsupportedOperationException</code>.</p>  
194 \*  
195 \* <p>I'm not sure the intention of the original authors for this method, but it seems to  
196 \* me that readable/writable should be attributes of the <code>DynaProperty</code> class  
197 \* (which they are not) and is the reason this method has not been implemented.</p>  
198 \*  
199 \* @param name Name of the new dynamic property  
200 \* @param type Data type of the new dynamic property (null for no  
201 \* restrictions)  
202 \* @param readable Set to <code>true</code> if this property value  
203 \* should be readable  
204 \* @param writeable Set to <code>true</code> if this property value  
205 \* should be writeable  
206 \*  
207 \* @throws UnsupportedOperationException anytime this method is called  
208 \*/  
209 public void add(final String name, final Class<?> type, final boolean readable, final boolean writeable) {  
210 throw new java.lang.UnsupportedOperationException("readable/writable properties not supported");  
211 }  
212  
213 /\*\*  
214 \* Add a new dynamic property.  
215 \*  
216 \* @param property Property the new dynamic property to add.  
217 \*  
218 \* @throws IllegalArgumentException if name is null  
219 \* @throws IllegalStateException if this DynaClass is currently  
220 \* restricted, so no new properties can be added  
221 \*/  
222 protected void add(final DynaProperty property) {  
223  
224 if (property.getName() == null) {  
225 throw new IllegalArgumentException("Property name is missing.");  
226 }  
227  
228 if (isRestricted()) {  
229 throw new IllegalStateException("DynaClass is currently restricted. No new properties can be added.");  
230 }  
231  
232 // Check if property already exists  
233 if (propertiesMap.get(property.getName()) != null) {  
234 return;  
235 }  
236  
237 // Create a new property array with the specified property  
238 final DynaProperty[] oldProperties = getDynaProperties();  
239 final DynaProperty[] newProperties = new DynaProperty[oldProperties.length+1];  
240 System.arraycopy(oldProperties, 0, newProperties, 0, oldProperties.length);  
241 newProperties[oldProperties.length] = property;  
242  
243 // Update the properties  
244 setProperties(newProperties);  
245  
246 }  
247  
248 /\*\*  
249 \* Remove the specified dynamic property, and any associated data type,  
250 \* readability, and writeability, from this dynamic class.  
251 \* <strong>NOTE</strong> - This does <strong>NOT</strong> cause any  
252 \* corresponding property values to be removed from DynaBean instances  
253 \* associated with this DynaClass.  
254 \*  
255 \* @param name Name of the dynamic property to remove  
256 \*  
257 \* @throws IllegalArgumentException if name is null  
258 \* @throws IllegalStateException if this DynaClass is currently  
259 \* restricted, so no properties can be removed  
260 \*/  
261 public void remove(final String name) {  
262  
263 if (name == null) {  
264 throw new IllegalArgumentException("Property name is missing.");  
265 }  
266  
267 if (isRestricted()) {  
268 throw new IllegalStateException("DynaClass is currently restricted. No properties can be removed.");  
269 }  
270  
271 // Ignore if property doesn't exist  
272 if (propertiesMap.get(name) == null) {  
273 return;  
274 }  
275  
276  
277 // Create a new property array of without the specified property  
278 final DynaProperty[] oldProperties = getDynaProperties();  
279 final DynaProperty[] newProperties = new DynaProperty[oldProperties.length-1];  
280 int j = 0;  
281 for (int i = 0; i < oldProperties.length; i++) {  
282 if (!(name.equals(oldProperties[i].getName()))) {  
283 newProperties[j] = oldProperties[i];  
284 j++;  
285 }  
286 }  
287  
288 // Update the properties  
289 setProperties(newProperties);  
290  
291 }  
292  
293 /\*\*  
294 \* <p>Return a property descriptor for the specified property.</p>  
295 \*  
296 \* <p>If the property is not found and the <code>returnNull</code> indicator is  
297 \* <code>true</code>, this method always returns <code>null</code>.</p>  
298 \*  
299 \* <p>If the property is not found and the <code>returnNull</code> indicator is  
300 \* <code>false</code> a new property descriptor is created and returned (although  
301 \* its not actually added to the DynaClass's properties). This is the default  
302 \* beahviour.</p>  
303 \*  
304 \* <p>The reason for not returning a <code>null</code> property descriptor is that  
305 \* <code>BeanUtils</code> uses this method to check if a property exists  
306 \* before trying to set it - since these <i>Lazy</i> implementations automatically  
307 \* add any new properties when they are set, returning <code>null</code> from  
308 \* this method would defeat their purpose.</p>  
309 \*  
310 \* @param name Name of the dynamic property for which a descriptor  
311 \* is requested  
312 \* @return The dyna property for the specified name  
313 \*  
314 \* @throws IllegalArgumentException if no property name is specified  
315 \*/  
316 @Override  
317 public DynaProperty getDynaProperty(final String name) {  
318  
319 if (name == null) {  
320 throw new IllegalArgumentException("Property name is missing.");  
321 }  
322  
323 DynaProperty dynaProperty = propertiesMap.get(name);  
324  
325 // If it doesn't exist and returnNull is false  
326 // create a new DynaProperty  
327 if (dynaProperty == null && !isReturnNull() && !isRestricted()) {  
328 dynaProperty = new DynaProperty(name);  
329 }  
330  
331 return dynaProperty;  
332  
333 }  
334  
335 /\*\*  
336 \* <p>Indicate whether a property actually exists.</p>  
337 \*  
338 \* <p><strong>N.B.</strong> Using <code>getDynaProperty(name) == null</code>  
339 \* doesn't work in this implementation because that method might  
340 \* return a DynaProperty if it doesn't exist (depending on the  
341 \* <code>returnNull</code> indicator).</p>  
342 \*  
343 \* @param name The name of the property to check  
344 \* @return <code>true</code> if there is a property of the  
345 \* specified name, otherwise <code>false</code>  
346 \* @throws IllegalArgumentException if no property name is specified  
347 \*/  
348 public boolean isDynaProperty(final String name) {  
349  
350 if (name == null) {  
351 throw new IllegalArgumentException("Property name is missing.");  
352 }  
353  
354 return propertiesMap.get(name) == null ? false : true;  
355  
356 }  
357  
358}