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  
019import java.util.Map;  
020  
021/\*\*  
022 \* <p>Provides a <i>light weight</i> <code>DynaBean</code> facade to a <code>Map</code>  
023 \* with <i>lazy</i> map/list processing.</p>  
024 \*  
025 \* <p>Its a <i>light weight</i> <code>DynaBean</code> implementation because there is no  
026 \* actual <code>DynaClass</code> associated with this <code>DynaBean</code> - in fact  
027 \* it implements the <code>DynaClass</code> interface itself providing <i>pseudo</i> DynaClass  
028 \* behaviour from the actual values stored in the <code>Map</code>.</p>  
029 \*  
030 \* <p>As well providing rhe standard <code>DynaBean</code> access to the <code>Map</code>'s properties  
031 \* this class also provides the usual <i>Lazy</i> behaviour:</p>  
032 \* <ul>  
033 \* <li>Properties don't need to be pre-defined in a <code>DynaClass</code></li>  
034 \* <li>Indexed properties (<code>Lists</code> or <code>Arrays</code>) are automatically instantiated  
035 \* and <i>grown</i> so that they are large enough to cater for the index being set.</li>  
036 \* <li>Mapped properties are automatically instantiated.</li>  
037 \* </ul>  
038 \*  
039 \* <p><b><u><i>Restricted</i> DynaClass</u></b></p>  
040 \* <p>This class implements the <code>MutableDynaClass</code> interface.  
041 \* <code>MutableDynaClass</code> have a facility to <i>restrict</i> the <code>DynaClass</code>  
042 \* so that its properties cannot be modified. If the <code>MutableDynaClass</code> is  
043 \* restricted then calling any of the <code>set()</code> methods for a property which  
044 \* doesn't exist will result in a <code>IllegalArgumentException</code> being thrown.</p>  
045 \*  
046 \* @version $Id$  
047 \*/  
048public class LazyDynaMap extends LazyDynaBean implements MutableDynaClass {  
049  
050 /\*\*  
051 \* The name of this DynaClass (analogous to the  
052 \* <code>getName()</code> method of <code>java.lang.Class</code>).  
053 \*/  
054 protected String name;  
055  
056 /\*\*  
057 \* Controls whether changes to this DynaClass's properties are allowed.  
058 \*/  
059 protected boolean restricted;  
060  
061 /\*\*  
062 \* <p>Controls whether the <code>getDynaProperty()</code> method returns  
063 \* null if a property doesn't exist - or creates a new one.</p>  
064 \*  
065 \* <p>Default is <code>false</code>.  
066 \*/  
067 protected boolean returnNull = false;  
068  
069  
070 // ------------------- Constructors ----------------------------------  
071  
072 /\*\*  
073 \* Default Constructor.  
074 \*/  
075 public LazyDynaMap() {  
076 this(null, (Map<String, Object>)null);  
077 }  
078  
079 /\*\*  
080 \* Construct a new <code>LazyDynaMap</code> with the specified name.  
081 \*  
082 \* @param name Name of this DynaBean class  
083 \*/  
084 public LazyDynaMap(final String name) {  
085 this(name, (Map<String, Object>)null);  
086 }  
087  
088 /\*\*  
089 \* Construct a new <code>LazyDynaMap</code> with the specified <code>Map</code>.  
090 \*  
091 \* @param values The Map backing this <code>LazyDynaMap</code>  
092 \*/  
093 public LazyDynaMap(final Map<String, Object> values) {  
094 this(null, values);  
095 }  
096  
097 /\*\*  
098 \* Construct a new <code>LazyDynaMap</code> with the specified name and <code>Map</code>.  
099 \*  
100 \* @param name Name of this DynaBean class  
101 \* @param values The Map backing this <code>LazyDynaMap</code>  
102 \*/  
103 public LazyDynaMap(final String name, final Map<String, Object> values) {  
104 this.name = name == null ? "LazyDynaMap" : name;  
105 this.values = values == null ? newMap() : values;  
106 this.dynaClass = this;  
107 }  
108  
109 /\*\*  
110 \* Construct a new <code>LazyDynaMap</code> with the specified properties.  
111 \*  
112 \* @param properties Property descriptors for the supported properties  
113 \*/  
114 public LazyDynaMap(final DynaProperty[] properties) {  
115 this(null, properties);  
116 }  
117  
118 /\*\*  
119 \* Construct a new <code>LazyDynaMap</code> with the specified name and properties.  
120 \*  
121 \* @param name Name of this DynaBean class  
122 \* @param properties Property descriptors for the supported properties  
123 \*/  
124 public LazyDynaMap(final String name, final DynaProperty[] properties) {  
125 this(name, (Map<String, Object>)null);  
126 if (properties != null) {  
127 for (DynaProperty propertie : properties) {  
128 add(propertie);  
129 }  
130 }  
131 }  
132  
133 /\*\*  
134 \* Construct a new <code>LazyDynaMap</code> based on an exisiting DynaClass  
135 \*  
136 \* @param dynaClass DynaClass to copy the name and properties from  
137 \*/  
138 public LazyDynaMap(final DynaClass dynaClass) {  
139 this(dynaClass.getName(), dynaClass.getDynaProperties());  
140 }  
141  
142 // ------------------- Public Methods ----------------------------------  
143  
144 /\*\*  
145 \* Set the Map backing this <code>DynaBean</code>  
146 \*  
147 \* @param values The new Map of values  
148 \*/  
149 public void setMap(final Map<String, Object> values) {  
150 this.values = values;  
151 }  
152  
153 /\*\*  
154 \* Return the underlying Map backing this <code>DynaBean</code>  
155 \* @return the underlying Map  
156 \* @since 1.8.0  
157 \*/  
158 @Override  
159 public Map<String, Object> getMap() {  
160 return values;  
161 }  
162  
163 // ------------------- DynaBean Methods ----------------------------------  
164  
165 /\*\*  
166 \* Set the value of a simple property with the specified name.  
167 \*  
168 \* @param name Name of the property whose value is to be set  
169 \* @param value Value to which this property is to be set  
170 \*/  
171 @Override  
172 public void set(final String name, final Object value) {  
173  
174 if (isRestricted() && !values.containsKey(name)) {  
175 throw new IllegalArgumentException  
176 ("Invalid property name '" + name + "' (DynaClass is restricted)");  
177 }  
178  
179 values.put(name, value);  
180  
181 }  
182  
183 // ------------------- DynaClass Methods ----------------------------------  
184  
185 /\*\*  
186 \* Return the name of this DynaClass (analogous to the  
187 \* <code>getName()</code> method of <code>java.lang.Class</code>)  
188 \*  
189 \* @return the name of the DynaClass  
190 \*/  
191 public String getName() {  
192 return this.name;  
193 }  
194  
195 /\*\*  
196 \* <p>Return a property descriptor for the specified property.</p>  
197 \*  
198 \* <p>If the property is not found and the <code>returnNull</code> indicator is  
199 \* <code>true</code>, this method always returns <code>null</code>.</p>  
200 \*  
201 \* <p>If the property is not found and the <code>returnNull</code> indicator is  
202 \* <code>false</code> a new property descriptor is created and returned (although  
203 \* its not actually added to the DynaClass's properties). This is the default  
204 \* beahviour.</p>  
205 \*  
206 \* <p>The reason for not returning a <code>null</code> property descriptor is that  
207 \* <code>BeanUtils</code> uses this method to check if a property exists  
208 \* before trying to set it - since these <i>Map</i> implementations automatically  
209 \* add any new properties when they are set, returning <code>null</code> from  
210 \* this method would defeat their purpose.</p>  
211 \*  
212 \* @param name Name of the dynamic property for which a descriptor  
213 \* is requested  
214 \* @return The descriptor for the specified property  
215 \*  
216 \* @throws IllegalArgumentException if no property name is specified  
217 \*/  
218 public DynaProperty getDynaProperty(final String name) {  
219  
220 if (name == null) {  
221 throw new IllegalArgumentException("Property name is missing.");  
222 }  
223  
224 // If it doesn't exist and returnNull is false  
225 // create a new DynaProperty  
226 if (!values.containsKey(name) && isReturnNull()) {  
227 return null;  
228 }  
229  
230 final Object value = values.get(name);  
231  
232 if (value == null) {  
233 return new DynaProperty(name);  
234 } else {  
235 return new DynaProperty(name, value.getClass());  
236 }  
237  
238 }  
239  
240 /\*\*  
241 \* <p>Return an array of <code>ProperyDescriptors</code> for the properties  
242 \* currently defined in this DynaClass. If no properties are defined, a  
243 \* zero-length array will be returned.</p>  
244 \*  
245 \* <p><strong>FIXME</strong> - Should we really be implementing  
246 \* <code>getBeanInfo()</code> instead, which returns property descriptors  
247 \* and a bunch of other stuff?</p>  
248 \* @return the set of properties for this DynaClass  
249 \*/  
250 public DynaProperty[] getDynaProperties() {  
251  
252 int i = 0;  
253 final DynaProperty[] properties = new DynaProperty[values.size()];  
254 for (final Map.Entry<String, Object> e : values.entrySet()) {  
255 final String name = e.getKey();  
256 final Object value = values.get(name);  
257 properties[i++] = new DynaProperty(name, value == null ? null  
258 : value.getClass());  
259 }  
260  
261 return properties;  
262  
263 }  
264  
265 /\*\*  
266 \* Instantiate and return a new DynaBean instance, associated  
267 \* with this DynaClass.  
268 \* @return A new <code>DynaBean</code> instance  
269 \*/  
270 public DynaBean newInstance() {  
271  
272 // Create a new instance of the Map  
273 Map<String, Object> newMap = null;  
274 try {  
275 @SuppressWarnings("unchecked")  
276 final  
277 // The new map is used as properties map  
278 Map<String, Object> temp = getMap().getClass().newInstance();  
279 newMap = temp;  
280 } catch(final Exception ex) {  
281 newMap = newMap();  
282 }  
283  
284 // Crate new LazyDynaMap and initialize properties  
285 final LazyDynaMap lazyMap = new LazyDynaMap(newMap);  
286 final DynaProperty[] properties = this.getDynaProperties();  
287 if (properties != null) {  
288 for (DynaProperty propertie : properties) {  
289 lazyMap.add(propertie);  
290 }  
291 }  
292 return lazyMap;  
293 }  
294  
295  
296 // ------------------- MutableDynaClass Methods ----------------------------------  
297  
298 /\*\*  
299 \* <p>Is this DynaClass currently restricted.</p>  
300 \* <p>If restricted, no changes to the existing registration of  
301 \* property names, data types, readability, or writeability are allowed.</p>  
302 \*  
303 \* @return <code>true</code> if this Mutable {@link DynaClass} is restricted,  
304 \* otherwise <code>false</code>  
305 \*/  
306 public boolean isRestricted() {  
307 return restricted;  
308 }  
309  
310 /\*\*  
311 \* <p>Set whether this DynaClass is currently restricted.</p>  
312 \* <p>If restricted, no changes to the existing registration of  
313 \* property names, data types, readability, or writeability are allowed.</p>  
314 \*  
315 \* @param restricted The new restricted state  
316 \*/  
317 public void setRestricted(final boolean restricted) {  
318 this.restricted = restricted;  
319 }  
320  
321 /\*\*  
322 \* Add a new dynamic property with no restrictions on data type,  
323 \* readability, or writeability.  
324 \*  
325 \* @param name Name of the new dynamic property  
326 \*  
327 \* @throws IllegalArgumentException if name is null  
328 \*/  
329 public void add(final String name) {  
330 add(name, null);  
331 }  
332  
333 /\*\*  
334 \* Add a new dynamic property with the specified data type, but with  
335 \* no restrictions on readability or writeability.  
336 \*  
337 \* @param name Name of the new dynamic property  
338 \* @param type Data type of the new dynamic property (null for no  
339 \* restrictions)  
340 \*  
341 \* @throws IllegalArgumentException if name is null  
342 \* @throws IllegalStateException if this DynaClass is currently  
343 \* restricted, so no new properties can be added  
344 \*/  
345 public void add(final String name, final Class<?> type) {  
346  
347 if (name == null) {  
348 throw new IllegalArgumentException("Property name is missing.");  
349 }  
350  
351 if (isRestricted()) {  
352 throw new IllegalStateException("DynaClass is currently restricted. No new properties can be added.");  
353 }  
354  
355 final Object value = values.get(name);  
356  
357 // Check if the property already exists  
358 if (value == null) {  
359 values.put(name, type == null ? null : createProperty(name, type));  
360 }  
361  
362 }  
363  
364 /\*\*  
365 \* <p>Add a new dynamic property with the specified data type, readability,  
366 \* and writeability.</p>  
367 \*  
368 \* <p><strong>N.B.</strong>Support for readable/writeable properties has not been implemented  
369 \* and this method always throws a <code>UnsupportedOperationException</code>.</p>  
370 \*  
371 \* <p>I'm not sure the intention of the original authors for this method, but it seems to  
372 \* me that readable/writable should be attributes of the <code>DynaProperty</code> class  
373 \* (which they are not) and is the reason this method has not been implemented.</p>  
374 \*  
375 \* @param name Name of the new dynamic property  
376 \* @param type Data type of the new dynamic property (null for no  
377 \* restrictions)  
378 \* @param readable Set to <code>true</code> if this property value  
379 \* should be readable  
380 \* @param writeable Set to <code>true</code> if this property value  
381 \* should be writeable  
382 \*  
383 \* @throws UnsupportedOperationException anytime this method is called  
384 \*/  
385 public void add(final String name, final Class<?> type, final boolean readable, final boolean writeable) {  
386 throw new java.lang.UnsupportedOperationException("readable/writable properties not supported");  
387 }  
388  
389 /\*\*  
390 \* Add a new dynamic property.  
391 \*  
392 \* @param property Property the new dynamic property to add.  
393 \*  
394 \* @throws IllegalArgumentException if name is null  
395 \*/  
396 protected void add(final DynaProperty property) {  
397 add(property.getName(), property.getType());  
398 }  
399  
400 /\*\*  
401 \* Remove the specified dynamic property, and any associated data type,  
402 \* readability, and writeability, from this dynamic class.  
403 \* <strong>NOTE</strong> - This does <strong>NOT</strong> cause any  
404 \* corresponding property values to be removed from DynaBean instances  
405 \* associated with this DynaClass.  
406 \*  
407 \* @param name Name of the dynamic property to remove  
408 \*  
409 \* @throws IllegalArgumentException if name is null  
410 \* @throws IllegalStateException if this DynaClass is currently  
411 \* restricted, so no properties can be removed  
412 \*/  
413 public void remove(final String name) {  
414  
415 if (name == null) {  
416 throw new IllegalArgumentException("Property name is missing.");  
417 }  
418  
419 if (isRestricted()) {  
420 throw new IllegalStateException("DynaClass is currently restricted. No properties can be removed.");  
421 }  
422  
423 // Remove, if property doesn't exist  
424 if (values.containsKey(name)) {  
425 values.remove(name);  
426 }  
427  
428 }  
429  
430  
431 // ------------------- Additional Public Methods ----------------------------------  
432  
433 /\*\*  
434 \* Should this DynaClass return a <code>null</code> from  
435 \* the <code>getDynaProperty(name)</code> method if the property  
436 \* doesn't exist.  
437 \*  
438 \* @return <code>true</code> if a <code>null</code> {@link DynaProperty}  
439 \* should be returned if the property doesn't exist, otherwise  
440 \* <code>false</code> if a new {@link DynaProperty} should be created.  
441 \*/  
442 public boolean isReturnNull() {  
443 return returnNull;  
444 }  
445  
446 /\*\*  
447 \* Set whether this DynaClass should return a <code>null</code> from  
448 \* the <code>getDynaProperty(name)</code> method if the property  
449 \* doesn't exist.  
450 \*  
451 \* @param returnNull <code>true</code> if a <code>null</code> {@link DynaProperty}  
452 \* should be returned if the property doesn't exist, otherwise  
453 \* <code>false</code> if a new {@link DynaProperty} should be created.  
454 \*/  
455 public void setReturnNull(final boolean returnNull) {  
456 this.returnNull = returnNull;  
457 }  
458  
459  
460 // ------------------- Protected Methods ----------------------------------  
461  
462 /\*\*  
463 \* <p>Indicate whether a property actually exists.</p>  
464 \*  
465 \* <p><strong>N.B.</strong> Using <code>getDynaProperty(name) == null</code>  
466 \* doesn't work in this implementation because that method might  
467 \* return a DynaProperty if it doesn't exist (depending on the  
468 \* <code>returnNull</code> indicator).</p>  
469 \*  
470 \* @param name Name of the dynamic property  
471 \* @return <code>true</code> if the property exists,  
472 \* otherwise <code>false</code>  
473 \* @throws IllegalArgumentException if no property name is specified  
474 \*/  
475 @Override  
476 protected boolean isDynaProperty(final String name) {  
477  
478 if (name == null) {  
479 throw new IllegalArgumentException("Property name is missing.");  
480 }  
481  
482 return values.containsKey(name);  
483  
484 }  
485  
486}