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.beans.PropertyDescriptor;  
022import java.lang.reflect.InvocationTargetException;  
023import java.lang.reflect.Method;  
024import java.util.Map;  
025  
026import org.apache.commons.collections.FastHashMap;  
027  
028  
029/\*\*  
030 \* <p>Utility methods for using Java Reflection APIs to facilitate generic  
031 \* property getter and setter operations on Java objects.</p>  
032 \*  
033 \* <p>The implementations for these methods are provided by <code>PropertyUtilsBean</code>.  
034 \* For more details see {@link PropertyUtilsBean}.</p>  
035 \*  
036 \* @version $Id$  
037 \* @see PropertyUtilsBean  
038 \* @see org.apache.commons.beanutils.expression.Resolver  
039 \*/  
040  
041public class PropertyUtils {  
042  
043  
044 // ----------------------------------------------------- Manifest Constants  
045  
046  
047 /\*\*  
048 \* The delimiter that preceeds the zero-relative subscript for an  
049 \* indexed reference.  
050 \*  
051 \* @deprecated The notation used for property name expressions is now  
052 \* dependant on the {@link org.apache.commons.beanutils.expression.Resolver}  
053 \* implementation being used.  
054 \*/  
055 @Deprecated  
056 public static final char INDEXED\_DELIM = '[';  
057  
058  
059 /\*\*  
060 \* The delimiter that follows the zero-relative subscript for an  
061 \* indexed reference.  
062 \*  
063 \* @deprecated The notation used for property name expressions is now  
064 \* dependant on the {@link org.apache.commons.beanutils.expression.Resolver}  
065 \* implementation being used.  
066 \*/  
067 @Deprecated  
068 public static final char INDEXED\_DELIM2 = ']';  
069  
070  
071 /\*\*  
072 \* The delimiter that preceeds the key of a mapped property.  
073 \*  
074 \* @deprecated The notation used for property name expressions is now  
075 \* dependant on the {@link org.apache.commons.beanutils.expression.Resolver}  
076 \* implementation being used.  
077 \*/  
078 @Deprecated  
079 public static final char MAPPED\_DELIM = '(';  
080  
081  
082 /\*\*  
083 \* The delimiter that follows the key of a mapped property.  
084 \*  
085 \* @deprecated The notation used for property name expressions is now  
086 \* dependant on the {@link org.apache.commons.beanutils.expression.Resolver}  
087 \* implementation being used.  
088 \*/  
089 @Deprecated  
090 public static final char MAPPED\_DELIM2 = ')';  
091  
092  
093 /\*\*  
094 \* The delimiter that separates the components of a nested reference.  
095 \*  
096 \* @deprecated The notation used for property name expressions is now  
097 \* dependant on the {@link org.apache.commons.beanutils.expression.Resolver}  
098 \* implementation being used.  
099 \*/  
100 @Deprecated  
101 public static final char NESTED\_DELIM = '.';  
102  
103  
104 // ------------------------------------------------------- Static Variables  
105  
106  
107 /\*\*  
108 \* The debugging detail level for this component.  
109 \*  
110 \* Note that this static variable will have unexpected side-effects if  
111 \* this class is deployed in a shared classloader within a container.  
112 \* However as it is actually completely ignored by this class due to its  
113 \* deprecated status, it doesn't do any actual harm.  
114 \*  
115 \* @deprecated The <code>debug</code> static property is no longer used  
116 \*/  
117 @Deprecated  
118 private static int debug = 0;  
119  
120 /\*\*  
121 \* The <code>debug</code> static property is no longer used  
122 \* @return debug property  
123 \* @deprecated The <code>debug</code> static property is no longer used  
124 \*/  
125 @Deprecated  
126 public static int getDebug() {  
127 return (debug);  
128 }  
129  
130 /\*\*  
131 \* The <code>debug</code> static property is no longer used  
132 \* @param newDebug debug property  
133 \* @deprecated The <code>debug</code> static property is no longer used  
134 \*/  
135 @Deprecated  
136 public static void setDebug(final int newDebug) {  
137 debug = newDebug;  
138 }  
139  
140 // --------------------------------------------------------- Public Methods  
141  
142  
143 /\*\*  
144 \* Clear any cached property descriptors information for all classes  
145 \* loaded by any class loaders. This is useful in cases where class  
146 \* loaders are thrown away to implement class reloading.  
147 \*  
148 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
149 \*  
150 \* @see PropertyUtilsBean#clearDescriptors  
151 \*/  
152 public static void clearDescriptors() {  
153  
154 PropertyUtilsBean.getInstance().clearDescriptors();  
155  
156 }  
157  
158 /\*\*  
159 \* Resets the registered {@link BeanIntrospector} objects to the initial default  
160 \* state.  
161 \*  
162 \* @since 1.9  
163 \*/  
164 public static void resetBeanIntrospectors() {  
165 PropertyUtilsBean.getInstance().resetBeanIntrospectors();  
166 }  
167  
168 /\*\*  
169 \* Adds a <code>BeanIntrospector</code>. This object is invoked when the  
170 \* property descriptors of a class need to be obtained.  
171 \*  
172 \* @param introspector the <code>BeanIntrospector</code> to be added (must  
173 \* not be <b>null</b>  
174 \* @throws IllegalArgumentException if the argument is <b>null</b>  
175 \* @since 1.9  
176 \*/  
177 public static void addBeanIntrospector(final BeanIntrospector introspector) {  
178 PropertyUtilsBean.getInstance().addBeanIntrospector(introspector);  
179 }  
180  
181 /\*\*  
182 \* Removes the specified <code>BeanIntrospector</code>.  
183 \*  
184 \* @param introspector the <code>BeanIntrospector</code> to be removed  
185 \* @return <b>true</b> if the <code>BeanIntrospector</code> existed and  
186 \* could be removed, <b>false</b> otherwise  
187 \* @since 1.9  
188 \*/  
189 public static boolean removeBeanIntrospector(final BeanIntrospector introspector) {  
190 return PropertyUtilsBean.getInstance().removeBeanIntrospector(  
191 introspector);  
192 }  
193  
194 /\*\*  
195 \* <p>Copy property values from the "origin" bean to the "destination" bean  
196 \* for all cases where the property names are the same (even though the  
197 \* actual getter and setter methods might have been customized via  
198 \* <code>BeanInfo</code> classes).</p>  
199 \*  
200 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
201 \*  
202 \* @param dest Destination bean whose properties are modified  
203 \* @param orig Origin bean whose properties are retrieved  
204 \*  
205 \* @throws IllegalAccessException if the caller does not have  
206 \* access to the property accessor method  
207 \* @throws IllegalArgumentException if the <code>dest</code> or  
208 \* <code>orig</code> argument is null  
209 \* @throws InvocationTargetException if the property accessor method  
210 \* throws an exception  
211 \* @throws NoSuchMethodException if an accessor method for this  
212 \* propety cannot be found  
213 \* @see PropertyUtilsBean#copyProperties  
214 \*/  
215 public static void copyProperties(final Object dest, final Object orig)  
216 throws IllegalAccessException, InvocationTargetException,  
217 NoSuchMethodException {  
218  
219 PropertyUtilsBean.getInstance().copyProperties(dest, orig);  
220 }  
221  
222  
223 /\*\*  
224 \* <p>Return the entire set of properties for which the specified bean  
225 \* provides a read method.</p>  
226 \*  
227 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
228 \*  
229 \* @param bean Bean whose properties are to be extracted  
230 \* @return The set of properties for the bean  
231 \*  
232 \* @throws IllegalAccessException if the caller does not have  
233 \* access to the property accessor method  
234 \* @throws IllegalArgumentException if <code>bean</code> is null  
235 \* @throws InvocationTargetException if the property accessor method  
236 \* throws an exception  
237 \* @throws NoSuchMethodException if an accessor method for this  
238 \* propety cannot be found  
239 \* @see PropertyUtilsBean#describe  
240 \*/  
241 public static Map<String, Object> describe(final Object bean)  
242 throws IllegalAccessException, InvocationTargetException,  
243 NoSuchMethodException {  
244  
245 return (PropertyUtilsBean.getInstance().describe(bean));  
246  
247 }  
248  
249  
250 /\*\*  
251 \* <p>Return the value of the specified indexed property of the specified  
252 \* bean, with no type conversions.</p>  
253 \*  
254 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
255 \*  
256 \* @param bean Bean whose property is to be extracted  
257 \* @param name <code>propertyname[index]</code> of the property value  
258 \* to be extracted  
259 \* @return the indexed property value  
260 \*  
261 \* @throws IndexOutOfBoundsException if the specified index  
262 \* is outside the valid range for the underlying property  
263 \* @throws IllegalAccessException if the caller does not have  
264 \* access to the property accessor method  
265 \* @throws IllegalArgumentException if <code>bean</code> or  
266 \* <code>name</code> is null  
267 \* @throws InvocationTargetException if the property accessor method  
268 \* throws an exception  
269 \* @throws NoSuchMethodException if an accessor method for this  
270 \* propety cannot be found  
271 \* @see PropertyUtilsBean#getIndexedProperty(Object,String)  
272 \*/  
273 public static Object getIndexedProperty(final Object bean, final String name)  
274 throws IllegalAccessException, InvocationTargetException,  
275 NoSuchMethodException {  
276  
277 return (PropertyUtilsBean.getInstance().getIndexedProperty(bean, name));  
278  
279 }  
280  
281  
282 /\*\*  
283 \* <p>Return the value of the specified indexed property of the specified  
284 \* bean, with no type conversions.</p>  
285 \*  
286 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
287 \*  
288 \* @param bean Bean whose property is to be extracted  
289 \* @param name Simple property name of the property value to be extracted  
290 \* @param index Index of the property value to be extracted  
291 \* @return the indexed property value  
292 \*  
293 \* @throws IndexOutOfBoundsException if the specified index  
294 \* is outside the valid range for the underlying property  
295 \* @throws IllegalAccessException if the caller does not have  
296 \* access to the property accessor method  
297 \* @throws IllegalArgumentException if <code>bean</code> or  
298 \* <code>name</code> is null  
299 \* @throws InvocationTargetException if the property accessor method  
300 \* throws an exception  
301 \* @throws NoSuchMethodException if an accessor method for this  
302 \* propety cannot be found  
303 \* @see PropertyUtilsBean#getIndexedProperty(Object,String, int)  
304 \*/  
305 public static Object getIndexedProperty(final Object bean,  
306 final String name, final int index)  
307 throws IllegalAccessException, InvocationTargetException,  
308 NoSuchMethodException {  
309  
310 return (PropertyUtilsBean.getInstance().getIndexedProperty(bean, name, index));  
311 }  
312  
313  
314 /\*\*  
315 \* <p>Return the value of the specified mapped property of the  
316 \* specified bean, with no type conversions.</p>  
317 \*  
318 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
319 \*  
320 \* @param bean Bean whose property is to be extracted  
321 \* @param name <code>propertyname(key)</code> of the property value  
322 \* to be extracted  
323 \* @return the mapped property value  
324 \*  
325 \* @throws IllegalAccessException if the caller does not have  
326 \* access to the property accessor method  
327 \* @throws InvocationTargetException if the property accessor method  
328 \* throws an exception  
329 \* @throws NoSuchMethodException if an accessor method for this  
330 \* propety cannot be found  
331 \* @see PropertyUtilsBean#getMappedProperty(Object,String)  
332 \*/  
333 public static Object getMappedProperty(final Object bean, final String name)  
334 throws IllegalAccessException, InvocationTargetException,  
335 NoSuchMethodException {  
336  
337 return (PropertyUtilsBean.getInstance().getMappedProperty(bean, name));  
338  
339 }  
340  
341  
342 /\*\*  
343 \* <p>Return the value of the specified mapped property of the specified  
344 \* bean, with no type conversions.</p>  
345 \*  
346 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
347 \*  
348 \* @param bean Bean whose property is to be extracted  
349 \* @param name Mapped property name of the property value to be extracted  
350 \* @param key Key of the property value to be extracted  
351 \* @return the mapped property value  
352 \*  
353 \* @throws IllegalAccessException if the caller does not have  
354 \* access to the property accessor method  
355 \* @throws InvocationTargetException if the property accessor method  
356 \* throws an exception  
357 \* @throws NoSuchMethodException if an accessor method for this  
358 \* propety cannot be found  
359 \* @see PropertyUtilsBean#getMappedProperty(Object,String, String)  
360 \*/  
361 public static Object getMappedProperty(final Object bean,  
362 final String name, final String key)  
363 throws IllegalAccessException, InvocationTargetException,  
364 NoSuchMethodException {  
365  
366 return PropertyUtilsBean.getInstance().getMappedProperty(bean, name, key);  
367  
368 }  
369  
370  
371 /\*\*  
372 \* <p>Return the mapped property descriptors for this bean class.</p>  
373 \*  
374 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
375 \*  
376 \* @param beanClass Bean class to be introspected  
377 \* @return the mapped property descriptors  
378 \* @see PropertyUtilsBean#getMappedPropertyDescriptors(Class)  
379 \* @deprecated This method should not be exposed  
380 \*/  
381 @Deprecated  
382 public static FastHashMap getMappedPropertyDescriptors(final Class<?> beanClass) {  
383  
384 return PropertyUtilsBean.getInstance().getMappedPropertyDescriptors(beanClass);  
385  
386 }  
387  
388  
389 /\*\*  
390 \* <p>Return the mapped property descriptors for this bean.</p>  
391 \*  
392 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
393 \*  
394 \* @param bean Bean to be introspected  
395 \* @return the mapped property descriptors  
396 \* @see PropertyUtilsBean#getMappedPropertyDescriptors(Object)  
397 \* @deprecated This method should not be exposed  
398 \*/  
399 @Deprecated  
400 public static FastHashMap getMappedPropertyDescriptors(final Object bean) {  
401  
402 return PropertyUtilsBean.getInstance().getMappedPropertyDescriptors(bean);  
403  
404 }  
405  
406  
407 /\*\*  
408 \* <p>Return the value of the (possibly nested) property of the specified  
409 \* name, for the specified bean, with no type conversions.</p>  
410 \*  
411 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
412 \*  
413 \* @param bean Bean whose property is to be extracted  
414 \* @param name Possibly nested name of the property to be extracted  
415 \* @return the nested property value  
416 \*  
417 \* @throws IllegalAccessException if the caller does not have  
418 \* access to the property accessor method  
419 \* @throws IllegalArgumentException if <code>bean</code> or  
420 \* <code>name</code> is null  
421 \* @throws NestedNullException if a nested reference to a  
422 \* property returns null  
423 \* @throws InvocationTargetException  
424 \* if the property accessor method throws an exception  
425 \* @throws NoSuchMethodException if an accessor method for this  
426 \* propety cannot be found  
427 \* @see PropertyUtilsBean#getNestedProperty  
428 \*/  
429 public static Object getNestedProperty(final Object bean, final String name)  
430 throws IllegalAccessException, InvocationTargetException,  
431 NoSuchMethodException {  
432  
433 return PropertyUtilsBean.getInstance().getNestedProperty(bean, name);  
434  
435 }  
436  
437  
438 /\*\*  
439 \* <p>Return the value of the specified property of the specified bean,  
440 \* no matter which property reference format is used, with no  
441 \* type conversions.</p>  
442 \*  
443 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
444 \*  
445 \* @param bean Bean whose property is to be extracted  
446 \* @param name Possibly indexed and/or nested name of the property  
447 \* to be extracted  
448 \* @return the property value  
449 \*  
450 \* @throws IllegalAccessException if the caller does not have  
451 \* access to the property accessor method  
452 \* @throws IllegalArgumentException if <code>bean</code> or  
453 \* <code>name</code> is null  
454 \* @throws InvocationTargetException if the property accessor method  
455 \* throws an exception  
456 \* @throws NoSuchMethodException if an accessor method for this  
457 \* propety cannot be found  
458 \* @see PropertyUtilsBean#getProperty  
459 \*/  
460 public static Object getProperty(final Object bean, final String name)  
461 throws IllegalAccessException, InvocationTargetException,  
462 NoSuchMethodException {  
463  
464 return (PropertyUtilsBean.getInstance().getProperty(bean, name));  
465  
466 }  
467  
468  
469 /\*\*  
470 \* <p>Retrieve the property descriptor for the specified property of the  
471 \* specified bean, or return <code>null</code> if there is no such  
472 \* descriptor.</p>  
473 \*  
474 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
475 \*  
476 \* @param bean Bean for which a property descriptor is requested  
477 \* @param name Possibly indexed and/or nested name of the property for  
478 \* which a property descriptor is requested  
479 \* @return the property descriptor  
480 \*  
481 \* @throws IllegalAccessException if the caller does not have  
482 \* access to the property accessor method  
483 \* @throws IllegalArgumentException if <code>bean</code> or  
484 \* <code>name</code> is null  
485 \* @throws IllegalArgumentException if a nested reference to a  
486 \* property returns null  
487 \* @throws InvocationTargetException if the property accessor method  
488 \* throws an exception  
489 \* @throws NoSuchMethodException if an accessor method for this  
490 \* propety cannot be found  
491 \* @see PropertyUtilsBean#getPropertyDescriptor  
492 \*/  
493 public static PropertyDescriptor getPropertyDescriptor(final Object bean,  
494 final String name)  
495 throws IllegalAccessException, InvocationTargetException,  
496 NoSuchMethodException {  
497  
498 return PropertyUtilsBean.getInstance().getPropertyDescriptor(bean, name);  
499  
500 }  
501  
502  
503 /\*\*  
504 \* <p>Retrieve the property descriptors for the specified class,  
505 \* introspecting and caching them the first time a particular bean class  
506 \* is encountered.</p>  
507 \*  
508 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
509 \*  
510 \* @param beanClass Bean class for which property descriptors are requested  
511 \* @return the property descriptors  
512 \* @throws IllegalArgumentException if <code>beanClass</code> is null  
513 \* @see PropertyUtilsBean#getPropertyDescriptors(Class)  
514 \*/  
515 public static PropertyDescriptor[]  
516 getPropertyDescriptors(final Class<?> beanClass) {  
517  
518 return PropertyUtilsBean.getInstance().getPropertyDescriptors(beanClass);  
519  
520 }  
521  
522  
523 /\*\*  
524 \* <p>Retrieve the property descriptors for the specified bean,  
525 \* introspecting and caching them the first time a particular bean class  
526 \* is encountered.</p>  
527 \*  
528 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
529 \*  
530 \* @param bean Bean for which property descriptors are requested  
531 \* @return the property descriptors  
532 \* @throws IllegalArgumentException if <code>bean</code> is null  
533 \* @see PropertyUtilsBean#getPropertyDescriptors(Object)  
534 \*/  
535 public static PropertyDescriptor[] getPropertyDescriptors(final Object bean) {  
536  
537 return PropertyUtilsBean.getInstance().getPropertyDescriptors(bean);  
538  
539 }  
540  
541  
542 /\*\*  
543 \* <p>Return the Java Class repesenting the property editor class that has  
544 \* been registered for this property (if any).</p>  
545 \*  
546 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
547 \*  
548 \* @param bean Bean for which a property descriptor is requested  
549 \* @param name Possibly indexed and/or nested name of the property for  
550 \* which a property descriptor is requested  
551 \* @return the property editor class  
552 \*  
553 \* @throws IllegalAccessException if the caller does not have  
554 \* access to the property accessor method  
555 \* @throws IllegalArgumentException if <code>bean</code> or  
556 \* <code>name</code> is null  
557 \* @throws IllegalArgumentException if a nested reference to a  
558 \* property returns null  
559 \* @throws InvocationTargetException if the property accessor method  
560 \* throws an exception  
561 \* @throws NoSuchMethodException if an accessor method for this  
562 \* propety cannot be found  
563 \* @see PropertyUtilsBean#getPropertyEditorClass(Object,String)  
564 \*/  
565 public static Class<?> getPropertyEditorClass(final Object bean, final String name)  
566 throws IllegalAccessException, InvocationTargetException,  
567 NoSuchMethodException {  
568  
569 return PropertyUtilsBean.getInstance().getPropertyEditorClass(bean, name);  
570  
571 }  
572  
573  
574 /\*\*  
575 \* <p>Return the Java Class representing the property type of the specified  
576 \* property, or <code>null</code> if there is no such property for the  
577 \* specified bean.</p>  
578 \*  
579 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
580 \*  
581 \* @param bean Bean for which a property descriptor is requested  
582 \* @param name Possibly indexed and/or nested name of the property for  
583 \* which a property descriptor is requested  
584 \* @return The property type  
585 \*  
586 \* @throws IllegalAccessException if the caller does not have  
587 \* access to the property accessor method  
588 \* @throws IllegalArgumentException if <code>bean</code> or  
589 \* <code>name</code> is null  
590 \* @throws IllegalArgumentException if a nested reference to a  
591 \* property returns null  
592 \* @throws InvocationTargetException if the property accessor method  
593 \* throws an exception  
594 \* @throws NoSuchMethodException if an accessor method for this  
595 \* propety cannot be found  
596 \* @see PropertyUtilsBean#getPropertyType(Object, String)  
597 \*/  
598 public static Class<?> getPropertyType(final Object bean, final String name)  
599 throws IllegalAccessException, InvocationTargetException,  
600 NoSuchMethodException {  
601  
602 return PropertyUtilsBean.getInstance().getPropertyType(bean, name);  
603 }  
604  
605  
606 /\*\*  
607 \* <p>Return an accessible property getter method for this property,  
608 \* if there is one; otherwise return <code>null</code>.</p>  
609 \*  
610 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
611 \*  
612 \* @param descriptor Property descriptor to return a getter for  
613 \* @return The read method  
614 \* @see PropertyUtilsBean#getReadMethod(PropertyDescriptor)  
615 \*/  
616 public static Method getReadMethod(final PropertyDescriptor descriptor) {  
617  
618 return (PropertyUtilsBean.getInstance().getReadMethod(descriptor));  
619  
620 }  
621  
622  
623 /\*\*  
624 \* <p>Return the value of the specified simple property of the specified  
625 \* bean, with no type conversions.</p>  
626 \*  
627 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
628 \*  
629 \* @param bean Bean whose property is to be extracted  
630 \* @param name Name of the property to be extracted  
631 \* @return The property value  
632 \*  
633 \* @throws IllegalAccessException if the caller does not have  
634 \* access to the property accessor method  
635 \* @throws IllegalArgumentException if <code>bean</code> or  
636 \* <code>name</code> is null  
637 \* @throws IllegalArgumentException if the property name  
638 \* is nested or indexed  
639 \* @throws InvocationTargetException if the property accessor method  
640 \* throws an exception  
641 \* @throws NoSuchMethodException if an accessor method for this  
642 \* propety cannot be found  
643 \* @see PropertyUtilsBean#getSimpleProperty  
644 \*/  
645 public static Object getSimpleProperty(final Object bean, final String name)  
646 throws IllegalAccessException, InvocationTargetException,  
647 NoSuchMethodException {  
648  
649 return PropertyUtilsBean.getInstance().getSimpleProperty(bean, name);  
650  
651 }  
652  
653  
654 /\*\*  
655 \* <p>Return an accessible property setter method for this property,  
656 \* if there is one; otherwise return <code>null</code>.</p>  
657 \*  
658 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
659 \*  
660 \* @param descriptor Property descriptor to return a setter for  
661 \* @return The write method  
662 \* @see PropertyUtilsBean#getWriteMethod(PropertyDescriptor)  
663 \*/  
664 public static Method getWriteMethod(final PropertyDescriptor descriptor) {  
665  
666 return PropertyUtilsBean.getInstance().getWriteMethod(descriptor);  
667  
668 }  
669  
670  
671 /\*\*  
672 \* <p>Return <code>true</code> if the specified property name identifies  
673 \* a readable property on the specified bean; otherwise, return  
674 \* <code>false</code>.</p>  
675 \*  
676 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
677 \*  
678 \* @param bean Bean to be examined (may be a {@link DynaBean}  
679 \* @param name Property name to be evaluated  
680 \* @return <code>true</code> if the property is readable,  
681 \* otherwise <code>false</code>  
682 \*  
683 \* @throws IllegalArgumentException if <code>bean</code>  
684 \* or <code>name</code> is <code>null</code>  
685 \* @see PropertyUtilsBean#isReadable  
686 \* @since BeanUtils 1.6  
687 \*/  
688 public static boolean isReadable(final Object bean, final String name) {  
689  
690 return PropertyUtilsBean.getInstance().isReadable(bean, name);  
691 }  
692  
693  
694 /\*\*  
695 \* <p>Return <code>true</code> if the specified property name identifies  
696 \* a writeable property on the specified bean; otherwise, return  
697 \* <code>false</code>.</p>  
698 \*  
699 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
700 \*  
701 \* @param bean Bean to be examined (may be a {@link DynaBean}  
702 \* @param name Property name to be evaluated  
703 \* @return <code>true</code> if the property is writeable,  
704 \* otherwise <code>false</code>  
705 \*  
706 \* @throws IllegalArgumentException if <code>bean</code>  
707 \* or <code>name</code> is <code>null</code>  
708 \* @see PropertyUtilsBean#isWriteable  
709 \* @since BeanUtils 1.6  
710 \*/  
711 public static boolean isWriteable(final Object bean, final String name) {  
712  
713 return PropertyUtilsBean.getInstance().isWriteable(bean, name);  
714 }  
715  
716  
717 /\*\*  
718 \* <p>Sets the value of the specified indexed property of the specified  
719 \* bean, with no type conversions.</p>  
720 \*  
721 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
722 \*  
723 \* @param bean Bean whose property is to be modified  
724 \* @param name <code>propertyname[index]</code> of the property value  
725 \* to be modified  
726 \* @param value Value to which the specified property element  
727 \* should be set  
728 \*  
729 \* @throws IndexOutOfBoundsException if the specified index  
730 \* is outside the valid range for the underlying property  
731 \* @throws IllegalAccessException if the caller does not have  
732 \* access to the property accessor method  
733 \* @throws IllegalArgumentException if <code>bean</code> or  
734 \* <code>name</code> is null  
735 \* @throws InvocationTargetException if the property accessor method  
736 \* throws an exception  
737 \* @throws NoSuchMethodException if an accessor method for this  
738 \* propety cannot be found  
739 \* @see PropertyUtilsBean#setIndexedProperty(Object, String, Object)  
740 \*/  
741 public static void setIndexedProperty(final Object bean, final String name,  
742 final Object value)  
743 throws IllegalAccessException, InvocationTargetException,  
744 NoSuchMethodException {  
745  
746 PropertyUtilsBean.getInstance().setIndexedProperty(bean, name, value);  
747  
748 }  
749  
750  
751 /\*\*  
752 \* <p>Sets the value of the specified indexed property of the specified  
753 \* bean, with no type conversions.</p>  
754 \*  
755 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
756 \*  
757 \* @param bean Bean whose property is to be set  
758 \* @param name Simple property name of the property value to be set  
759 \* @param index Index of the property value to be set  
760 \* @param value Value to which the indexed property element is to be set  
761 \*  
762 \* @throws IndexOutOfBoundsException if the specified index  
763 \* is outside the valid range for the underlying property  
764 \* @throws IllegalAccessException if the caller does not have  
765 \* access to the property accessor method  
766 \* @throws IllegalArgumentException if <code>bean</code> or  
767 \* <code>name</code> is null  
768 \* @throws InvocationTargetException if the property accessor method  
769 \* throws an exception  
770 \* @throws NoSuchMethodException if an accessor method for this  
771 \* propety cannot be found  
772 \* @see PropertyUtilsBean#setIndexedProperty(Object, String, Object)  
773 \*/  
774 public static void setIndexedProperty(final Object bean, final String name,  
775 final int index, final Object value)  
776 throws IllegalAccessException, InvocationTargetException,  
777 NoSuchMethodException {  
778  
779 PropertyUtilsBean.getInstance().setIndexedProperty(bean, name, index, value);  
780 }  
781  
782  
783 /\*\*  
784 \* <p>Sets the value of the specified mapped property of the  
785 \* specified bean, with no type conversions.</p>  
786 \*  
787 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
788 \*  
789 \* @param bean Bean whose property is to be set  
790 \* @param name <code>propertyname(key)</code> of the property value  
791 \* to be set  
792 \* @param value The property value to be set  
793 \*  
794 \* @throws IllegalAccessException if the caller does not have  
795 \* access to the property accessor method  
796 \* @throws InvocationTargetException if the property accessor method  
797 \* throws an exception  
798 \* @throws NoSuchMethodException if an accessor method for this  
799 \* propety cannot be found  
800 \* @see PropertyUtilsBean#setMappedProperty(Object, String, Object)  
801 \*/  
802 public static void setMappedProperty(final Object bean, final String name,  
803 final Object value)  
804 throws IllegalAccessException, InvocationTargetException,  
805 NoSuchMethodException {  
806  
807 PropertyUtilsBean.getInstance().setMappedProperty(bean, name, value);  
808 }  
809  
810  
811 /\*\*  
812 \* <p>Sets the value of the specified mapped property of the specified  
813 \* bean, with no type conversions.</p>  
814 \*  
815 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
816 \*  
817 \* @param bean Bean whose property is to be set  
818 \* @param name Mapped property name of the property value to be set  
819 \* @param key Key of the property value to be set  
820 \* @param value The property value to be set  
821 \*  
822 \* @throws IllegalAccessException if the caller does not have  
823 \* access to the property accessor method  
824 \* @throws InvocationTargetException if the property accessor method  
825 \* throws an exception  
826 \* @throws NoSuchMethodException if an accessor method for this  
827 \* propety cannot be found  
828 \* @see PropertyUtilsBean#setMappedProperty(Object, String, String, Object)  
829 \*/  
830 public static void setMappedProperty(final Object bean, final String name,  
831 final String key, final Object value)  
832 throws IllegalAccessException, InvocationTargetException,  
833 NoSuchMethodException {  
834  
835 PropertyUtilsBean.getInstance().setMappedProperty(bean, name, key, value);  
836 }  
837  
838  
839 /\*\*  
840 \* <p>Sets the value of the (possibly nested) property of the specified  
841 \* name, for the specified bean, with no type conversions.</p>  
842 \*  
843 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
844 \*  
845 \* @param bean Bean whose property is to be modified  
846 \* @param name Possibly nested name of the property to be modified  
847 \* @param value Value to which the property is to be set  
848 \*  
849 \* @throws IllegalAccessException if the caller does not have  
850 \* access to the property accessor method  
851 \* @throws IllegalArgumentException if <code>bean</code> or  
852 \* <code>name</code> is null  
853 \* @throws IllegalArgumentException if a nested reference to a  
854 \* property returns null  
855 \* @throws InvocationTargetException if the property accessor method  
856 \* throws an exception  
857 \* @throws NoSuchMethodException if an accessor method for this  
858 \* propety cannot be found  
859 \* @see PropertyUtilsBean#setNestedProperty  
860 \*/  
861 public static void setNestedProperty(final Object bean,  
862 final String name, final Object value)  
863 throws IllegalAccessException, InvocationTargetException,  
864 NoSuchMethodException {  
865  
866 PropertyUtilsBean.getInstance().setNestedProperty(bean, name, value);  
867 }  
868  
869  
870 /\*\*  
871 \* <p>Set the value of the specified property of the specified bean,  
872 \* no matter which property reference format is used, with no  
873 \* type conversions.</p>  
874 \*  
875 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
876 \*  
877 \* @param bean Bean whose property is to be modified  
878 \* @param name Possibly indexed and/or nested name of the property  
879 \* to be modified  
880 \* @param value Value to which this property is to be set  
881 \*  
882 \* @throws IllegalAccessException if the caller does not have  
883 \* access to the property accessor method  
884 \* @throws IllegalArgumentException if <code>bean</code> or  
885 \* <code>name</code> is null  
886 \* @throws InvocationTargetException if the property accessor method  
887 \* throws an exception  
888 \* @throws NoSuchMethodException if an accessor method for this  
889 \* propety cannot be found  
890 \* @see PropertyUtilsBean#setProperty  
891 \*/  
892 public static void setProperty(final Object bean, final String name, final Object value)  
893 throws IllegalAccessException, InvocationTargetException,  
894 NoSuchMethodException {  
895  
896 PropertyUtilsBean.getInstance().setProperty(bean, name, value);  
897  
898 }  
899  
900  
901 /\*\*  
902 \* <p>Set the value of the specified simple property of the specified bean,  
903 \* with no type conversions.</p>  
904 \*  
905 \* <p>For more details see <code>PropertyUtilsBean</code>.</p>  
906 \*  
907 \* @param bean Bean whose property is to be modified  
908 \* @param name Name of the property to be modified  
909 \* @param value Value to which the property should be set  
910 \*  
911 \* @throws IllegalAccessException if the caller does not have  
912 \* access to the property accessor method  
913 \* @throws IllegalArgumentException if <code>bean</code> or  
914 \* <code>name</code> is null  
915 \* @throws IllegalArgumentException if the property name is  
916 \* nested or indexed  
917 \* @throws InvocationTargetException if the property accessor method  
918 \* throws an exception  
919 \* @throws NoSuchMethodException if an accessor method for this  
920 \* propety cannot be found  
921 \* @see PropertyUtilsBean#setSimpleProperty  
922 \*/  
923 public static void setSimpleProperty(final Object bean,  
924 final String name, final Object value)  
925 throws IllegalAccessException, InvocationTargetException,  
926 NoSuchMethodException {  
927  
928 PropertyUtilsBean.getInstance().setSimpleProperty(bean, name, value);  
929 }  
930  
931  
932}