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  
021/\*\*  
022 \* <p>Utility methods for converting String scalar values to objects of the  
023 \* specified Class, String arrays to arrays of the specified Class.</p>  
024 \*  
025 \* <p>For more details, see <code>ConvertUtilsBean</code> which provides the  
026 \* implementations for these methods.</p>  
027 \*  
028 \* @version $Id$  
029 \* @see ConvertUtilsBean  
030 \*/  
031  
032public class ConvertUtils {  
033  
034  
035 // ------------------------------------------------------ Static Properties  
036  
037 /\*\*  
038 \* Gets the default value for Boolean conversions.  
039 \* @return The default Boolean value  
040 \* @deprecated Register replacement converters for Boolean.TYPE and  
041 \* Boolean.class instead  
042 \*/  
043 @Deprecated  
044 public static boolean getDefaultBoolean() {  
045 return (ConvertUtilsBean.getInstance().getDefaultBoolean());  
046 }  
047  
048 /\*\*  
049 \* Sets the default value for Boolean conversions.  
050 \* @param newDefaultBoolean The default Boolean value  
051 \* @deprecated Register replacement converters for Boolean.TYPE and  
052 \* Boolean.class instead  
053 \*/  
054 @Deprecated  
055 public static void setDefaultBoolean(final boolean newDefaultBoolean) {  
056 ConvertUtilsBean.getInstance().setDefaultBoolean(newDefaultBoolean);  
057 }  
058  
059  
060 /\*\*  
061 \* Gets the default value for Byte conversions.  
062 \* @return The default Byte value  
063 \* @deprecated Register replacement converters for Byte.TYPE and  
064 \* Byte.class instead  
065 \*/  
066 @Deprecated  
067 public static byte getDefaultByte() {  
068 return ConvertUtilsBean.getInstance().getDefaultByte();  
069 }  
070  
071 /\*\*  
072 \* Sets the default value for Byte conversions.  
073 \* @param newDefaultByte The default Byte value  
074 \* @deprecated Register replacement converters for Byte.TYPE and  
075 \* Byte.class instead  
076 \*/  
077 @Deprecated  
078 public static void setDefaultByte(final byte newDefaultByte) {  
079 ConvertUtilsBean.getInstance().setDefaultByte(newDefaultByte);  
080 }  
081  
082  
083 /\*\*  
084 \* Gets the default value for Character conversions.  
085 \* @return The default Character value  
086 \* @deprecated Register replacement converters for Character.TYPE and  
087 \* Character.class instead  
088 \*/  
089 @Deprecated  
090 public static char getDefaultCharacter() {  
091 return ConvertUtilsBean.getInstance().getDefaultCharacter();  
092 }  
093  
094 /\*\*  
095 \* Sets the default value for Character conversions.  
096 \* @param newDefaultCharacter The default Character value  
097 \* @deprecated Register replacement converters for Character.TYPE and  
098 \* Character.class instead  
099 \*/  
100 @Deprecated  
101 public static void setDefaultCharacter(final char newDefaultCharacter) {  
102 ConvertUtilsBean.getInstance().setDefaultCharacter(newDefaultCharacter);  
103 }  
104  
105  
106 /\*\*  
107 \* Gets the default value for Double conversions.  
108 \* @return The default Double value  
109 \* @deprecated Register replacement converters for Double.TYPE and  
110 \* Double.class instead  
111 \*/  
112 @Deprecated  
113 public static double getDefaultDouble() {  
114 return ConvertUtilsBean.getInstance().getDefaultDouble();  
115 }  
116  
117 /\*\*  
118 \* Sets the default value for Double conversions.  
119 \* @param newDefaultDouble The default Double value  
120 \* @deprecated Register replacement converters for Double.TYPE and  
121 \* Double.class instead  
122 \*/  
123 @Deprecated  
124 public static void setDefaultDouble(final double newDefaultDouble) {  
125 ConvertUtilsBean.getInstance().setDefaultDouble(newDefaultDouble);  
126 }  
127  
128  
129 /\*\*  
130 \* Get the default value for Float conversions.  
131 \* @return The default Float value  
132 \* @deprecated Register replacement converters for Float.TYPE and  
133 \* Float.class instead  
134 \*/  
135 @Deprecated  
136 public static float getDefaultFloat() {  
137 return ConvertUtilsBean.getInstance().getDefaultFloat();  
138 }  
139  
140 /\*\*  
141 \* Sets the default value for Float conversions.  
142 \* @param newDefaultFloat The default Float value  
143 \* @deprecated Register replacement converters for Float.TYPE and  
144 \* Float.class instead  
145 \*/  
146 @Deprecated  
147 public static void setDefaultFloat(final float newDefaultFloat) {  
148 ConvertUtilsBean.getInstance().setDefaultFloat(newDefaultFloat);  
149 }  
150  
151  
152 /\*\*  
153 \* Gets the default value for Integer conversions.  
154 \* @return The default Integer value  
155 \* @deprecated Register replacement converters for Integer.TYPE and  
156 \* Integer.class instead  
157 \*/  
158 @Deprecated  
159 public static int getDefaultInteger() {  
160 return ConvertUtilsBean.getInstance().getDefaultInteger();  
161 }  
162  
163 /\*\*  
164 \* Sets the default value for Integer conversions.  
165 \* @param newDefaultInteger The default Integer value  
166 \* @deprecated Register replacement converters for Integer.TYPE and  
167 \* Integer.class instead  
168 \*/  
169 @Deprecated  
170 public static void setDefaultInteger(final int newDefaultInteger) {  
171 ConvertUtilsBean.getInstance().setDefaultInteger(newDefaultInteger);  
172 }  
173  
174  
175 /\*\*  
176 \* Gets the default value for Long conversions.  
177 \* @return The default Long value  
178 \* @deprecated Register replacement converters for Long.TYPE and  
179 \* Long.class instead  
180 \*/  
181 @Deprecated  
182 public static long getDefaultLong() {  
183 return (ConvertUtilsBean.getInstance().getDefaultLong());  
184 }  
185  
186 /\*\*  
187 \* Sets the default value for Long conversions.  
188 \* @param newDefaultLong The default Long value  
189 \* @deprecated Register replacement converters for Long.TYPE and  
190 \* Long.class instead  
191 \*/  
192 @Deprecated  
193 public static void setDefaultLong(final long newDefaultLong) {  
194 ConvertUtilsBean.getInstance().setDefaultLong(newDefaultLong);  
195 }  
196  
197  
198 /\*\*  
199 \* Gets the default value for Short conversions.  
200 \* @return The default Short value  
201 \* @deprecated Register replacement converters for Short.TYPE and  
202 \* Short.class instead  
203 \*/  
204 @Deprecated  
205 public static short getDefaultShort() {  
206 return ConvertUtilsBean.getInstance().getDefaultShort();  
207 }  
208  
209 /\*\*  
210 \* Sets the default value for Short conversions.  
211 \* @param newDefaultShort The default Short value  
212 \* @deprecated Register replacement converters for Short.TYPE and  
213 \* Short.class instead  
214 \*/  
215 @Deprecated  
216 public static void setDefaultShort(final short newDefaultShort) {  
217 ConvertUtilsBean.getInstance().setDefaultShort(newDefaultShort);  
218 }  
219  
220 // --------------------------------------------------------- Public Classes  
221  
222  
223 /\*\*  
224 \* <p>Convert the specified value into a String.</p>  
225 \*  
226 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
227 \*  
228 \* @param value Value to be converted (may be null)  
229 \* @return The converted String value or null if value is null  
230 \*  
231 \* @see ConvertUtilsBean#convert(Object)  
232 \*/  
233 public static String convert(final Object value) {  
234 return ConvertUtilsBean.getInstance().convert(value);  
235 }  
236  
237  
238 /\*\*  
239 \* <p>Convert the specified value to an object of the specified class (if  
240 \* possible). Otherwise, return a String representation of the value.</p>  
241 \*  
242 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
243 \*  
244 \* @param value Value to be converted (may be null)  
245 \* @param clazz Java class to be converted to (must not be null)  
246 \* @return The converted value  
247 \*  
248 \* @see ConvertUtilsBean#convert(String, Class)  
249 \*/  
250 public static Object convert(final String value, final Class<?> clazz) {  
251 return ConvertUtilsBean.getInstance().convert(value, clazz);  
252 }  
253  
254  
255 /\*\*  
256 \* <p>Convert an array of specified values to an array of objects of the  
257 \* specified class (if possible).</p>  
258 \*  
259 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
260 \*  
261 \* @param values Array of values to be converted  
262 \* @param clazz Java array or element class to be converted to (must not be null)  
263 \* @return The converted value  
264 \*  
265 \* @see ConvertUtilsBean#convert(String[], Class)  
266 \*/  
267 public static Object convert(final String[] values, final Class<?> clazz) {  
268 return ConvertUtilsBean.getInstance().convert(values, clazz);  
269 }  
270  
271 /\*\*  
272 \* <p>Convert the value to an object of the specified class (if  
273 \* possible).</p>  
274 \*  
275 \* @param value Value to be converted (may be null)  
276 \* @param targetType Class of the value to be converted to (must not be null)  
277 \* @return The converted value  
278 \*  
279 \* @throws ConversionException if thrown by an underlying Converter  
280 \*/  
281 public static Object convert(final Object value, final Class<?> targetType) {  
282 return ConvertUtilsBean.getInstance().convert(value, targetType);  
283 }  
284  
285 /\*\*  
286 \* <p>Remove all registered {@link Converter}s, and re-establish the  
287 \* standard Converters.</p>  
288 \*  
289 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
290 \*  
291 \* @see ConvertUtilsBean#deregister()  
292 \*/  
293 public static void deregister() {  
294 ConvertUtilsBean.getInstance().deregister();  
295 }  
296  
297  
298 /\*\*  
299 \* <p>Remove any registered {@link Converter} for the specified destination  
300 \* <code>Class</code>.</p>  
301 \*  
302 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
303 \*  
304 \* @param clazz Class for which to remove a registered Converter  
305 \* @see ConvertUtilsBean#deregister(Class)  
306 \*/  
307 public static void deregister(final Class<?> clazz) {  
308 ConvertUtilsBean.getInstance().deregister(clazz);  
309 }  
310  
311  
312 /\*\*  
313 \* <p>Look up and return any registered {@link Converter} for the specified  
314 \* destination class; if there is no registered Converter, return  
315 \* <code>null</code>.</p>  
316 \*  
317 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
318 \*  
319 \* @param clazz Class for which to return a registered Converter  
320 \* @return The registered {@link Converter} or <code>null</code> if not found  
321 \* @see ConvertUtilsBean#lookup(Class)  
322 \*/  
323 public static Converter lookup(final Class<?> clazz) {  
324 return ConvertUtilsBean.getInstance().lookup(clazz);  
325 }  
326  
327 /\*\*  
328 \* Look up and return any registered {@link Converter} for the specified  
329 \* source and destination class; if there is no registered Converter,  
330 \* return <code>null</code>.  
331 \*  
332 \* @param sourceType Class of the value being converted  
333 \* @param targetType Class of the value to be converted to  
334 \* @return The registered {@link Converter} or <code>null</code> if not found  
335 \*/  
336 public static Converter lookup(final Class<?> sourceType, final Class<?> targetType) {  
337 return ConvertUtilsBean.getInstance().lookup(sourceType, targetType);  
338 }  
339  
340 /\*\*  
341 \* <p>Register a custom {@link Converter} for the specified destination  
342 \* <code>Class</code>, replacing any previously registered Converter.</p>  
343 \*  
344 \* <p>For more details see <code>ConvertUtilsBean</code>.</p>  
345 \*  
346 \* @param converter Converter to be registered  
347 \* @param clazz Destination class for conversions performed by this  
348 \* Converter  
349 \* @see ConvertUtilsBean#register(Converter, Class)  
350 \*/  
351 public static void register(final Converter converter, final Class<?> clazz) {  
352 ConvertUtilsBean.getInstance().register(converter, clazz);  
353 }  
354  
355  
356 /\*\*  
357 \* Change primitive Class types to the associated wrapper class. This is  
358 \* useful for concrete converter implementations which typically treat  
359 \* primitive types like their corresponding wrapper types.  
360 \*  
361 \* @param <T> The type to be checked.  
362 \* @param type The class type to check.  
363 \* @return The converted type.  
364 \* @since 1.9  
365 \*/  
366 // All type casts are safe because the TYPE members of the wrapper types  
367 // return their own class.  
368 @SuppressWarnings("unchecked")  
369 public static <T> Class<T> primitiveToWrapper(final Class<T> type) {  
370 if (type == null || !type.isPrimitive()) {  
371 return type;  
372 }  
373  
374 if (type == Integer.TYPE) {  
375 return (Class<T>) Integer.class;  
376 } else if (type == Double.TYPE) {  
377 return (Class<T>) Double.class;  
378 } else if (type == Long.TYPE) {  
379 return (Class<T>) Long.class;  
380 } else if (type == Boolean.TYPE) {  
381 return (Class<T>) Boolean.class;  
382 } else if (type == Float.TYPE) {  
383 return (Class<T>) Float.class;  
384 } else if (type == Short.TYPE) {  
385 return (Class<T>) Short.class;  
386 } else if (type == Byte.TYPE) {  
387 return (Class<T>) Byte.class;  
388 } else if (type == Character.TYPE) {  
389 return (Class<T>) Character.class;  
390 } else {  
391 return type;  
392 }  
393 }  
394}