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.converters;  
018  
019/\*\*  
020 \* {@link org.apache.commons.beanutils.Converter} implementation that handles conversion  
021 \* to and from <b>Boolean</b> objects.  
022 \* {@link org.apache.commons.beanutils.Converter} implementation that  
023 \* handles conversion to and from <b>java.lang.Boolean</b> objects.  
024 \* <p>  
025 \* Can be configured to either return a <i>default value</i> or throw a  
026 \* <code>ConversionException</code> if a conversion error occurs.  
027 \* <p>  
028 \* By default any object whose string representation is one of the values  
029 \* {"yes", "y", "true", "on", "1"} is converted to Boolean.TRUE, and  
030 \* string representations {"no", "n", "false", "off", "0"} are converted  
031 \* to Boolean.FALSE. The recognized true/false strings can be changed by:  
032 \* <pre>  
033 \* String[] trueStrings = {"oui", "o", "1"};  
034 \* String[] falseStrings = {"non", "n", "0"};  
035 \* Converter bc = new BooleanConverter(trueStrings, falseStrings);  
036 \* ConvertUtils.register(bc, Boolean.class);  
037 \* ConvertUtils.register(bc, Boolean.TYPE);  
038 \* </pre>  
039 \* In addition, it is recommended that the BooleanArrayConverter also be  
040 \* modified to recognise the same set of values:  
041 \* <pre>  
042 \* Converter bac = new BooleanArrayConverter(bc, BooleanArrayConverter.NO\_DEFAULT);  
043 \* ConvertUtils.register(bac, bac.MODEL);  
044 \* </pre>  
045 \* </p>  
046 \*  
047 \* <p>Case is ignored when converting values to true or false.</p>  
048 \*  
049 \* @version $Id$  
050 \* @since 1.3  
051 \*/  
052public final class BooleanConverter extends AbstractConverter {  
053  
054  
055 // ----------------------------------------------------------- Constructors  
056  
057  
058 /\*\*  
059 \* Create a {@link org.apache.commons.beanutils.Converter} that will throw a  
060 \* {@link org.apache.commons.beanutils.ConversionException}  
061 \* if a conversion error occurs, ie the string value being converted is  
062 \* not one of the known true strings, nor one of the known false strings.  
063 \*/  
064 public BooleanConverter() {  
065 super();  
066 }  
067  
068  
069 /\*\*  
070 \* Create a {@link org.apache.commons.beanutils.Converter} that will return the specified default value  
071 \* if a conversion error occurs, ie the string value being converted is  
072 \* not one of the known true strings, nor one of the known false strings.  
073 \*  
074 \* @param defaultValue The default value to be returned if the value  
075 \* being converted is not recognized. This value may be null, in which  
076 \* case null will be returned on conversion failure. When non-null, it is  
077 \* expected that this value will be either Boolean.TRUE or Boolean.FALSE.  
078 \* The special value BooleanConverter.NO\_DEFAULT can also be passed here,  
079 \* in which case this constructor acts like the no-argument one.  
080 \*/  
081 public BooleanConverter(final Object defaultValue) {  
082 super();  
083 if (defaultValue != NO\_DEFAULT) {  
084 setDefaultValue(defaultValue);  
085 }  
086 }  
087  
088 /\*\*  
089 \* Create a {@link org.apache.commons.beanutils.Converter} that will throw a  
090 \* {@link org.apache.commons.beanutils.ConversionException}  
091 \* if a conversion error occurs, ie the string value being converted is  
092 \* not one of the known true strings, nor one of the known false strings.  
093 \* <p>  
094 \* The provided string arrays are copied, so that changes to the elements  
095 \* of the array after this call is made do not affect this object.  
096 \*  
097 \* @param trueStrings is the set of strings which should convert to the  
098 \* value Boolean.TRUE. The value null must not be present. Case is  
099 \* ignored.  
100 \*  
101 \* @param falseStrings is the set of strings which should convert to the  
102 \* value Boolean.TRUE. The value null must not be present. Case is  
103 \* ignored.  
104 \* @since 1.8.0  
105 \*/  
106 public BooleanConverter(final String[] trueStrings, final String[] falseStrings) {  
107 super();  
108 this.trueStrings = copyStrings(trueStrings);  
109 this.falseStrings = copyStrings(falseStrings);  
110 }  
111  
112 /\*\*  
113 \* Create a {@link org.apache.commons.beanutils.Converter} that will return  
114 \* the specified default value if a conversion error occurs.  
115 \* <p>  
116 \* The provided string arrays are copied, so that changes to the elements  
117 \* of the array after this call is made do not affect this object.  
118 \*  
119 \* @param trueStrings is the set of strings which should convert to the  
120 \* value Boolean.TRUE. The value null must not be present. Case is  
121 \* ignored.  
122 \*  
123 \* @param falseStrings is the set of strings which should convert to the  
124 \* value Boolean.TRUE. The value null must not be present. Case is  
125 \* ignored.  
126 \*  
127 \* @param defaultValue The default value to be returned if the value  
128 \* being converted is not recognized. This value may be null, in which  
129 \* case null will be returned on conversion failure. When non-null, it is  
130 \* expected that this value will be either Boolean.TRUE or Boolean.FALSE.  
131 \* The special value BooleanConverter.NO\_DEFAULT can also be passed here,  
132 \* in which case an exception will be thrown on conversion failure.  
133 \* @since 1.8.0  
134 \*/  
135 public BooleanConverter(final String[] trueStrings, final String[] falseStrings,  
136 final Object defaultValue) {  
137 super();  
138 this.trueStrings = copyStrings(trueStrings);  
139 this.falseStrings = copyStrings(falseStrings);  
140 if (defaultValue != NO\_DEFAULT) {  
141 setDefaultValue(defaultValue);  
142 }  
143 }  
144  
145  
146 // ----------------------------------------------------- Static Variables  
147  
148  
149 /\*\*  
150 \* This is a special reference that can be passed as the "default object"  
151 \* to the constructor to indicate that no default is desired. Note that  
152 \* the value 'null' cannot be used for this purpose, as the caller may  
153 \* want a null to be returned as the default.  
154 \* @deprecated Use constructors without default value.  
155 \*/  
156 @Deprecated  
157 public static final Object NO\_DEFAULT = new Object();  
158  
159  
160 // ----------------------------------------------------- Instance Variables  
161  
162 /\*\*  
163 \* The set of strings that are known to map to Boolean.TRUE.  
164 \*/  
165 private String[] trueStrings = {"true", "yes", "y", "on", "1"};  
166  
167 /\*\*  
168 \* The set of strings that are known to map to Boolean.FALSE.  
169 \*/  
170 private String[] falseStrings = {"false", "no", "n", "off", "0"};  
171  
172 // --------------------------------------------------------- Protected Methods  
173  
174 /\*\*  
175 \* Return the default type this <code>Converter</code> handles.  
176 \*  
177 \* @return The default type this <code>Converter</code> handles.  
178 \* @since 1.8.0  
179 \*/  
180 @Override  
181 protected Class<Boolean> getDefaultType() {  
182 return Boolean.class;  
183 }  
184  
185 /\*\*  
186 \* Convert the specified input object into an output object of the  
187 \* specified type.  
188 \*  
189 \* @param <T> Target type of the conversion.  
190 \* @param type is the type to which this value should be converted. In the  
191 \* case of this BooleanConverter class, this value is ignored.  
192 \*  
193 \* @param value is the input value to be converted. The toString method  
194 \* shall be invoked on this object, and the result compared (ignoring  
195 \* case) against the known "true" and "false" string values.  
196 \*  
197 \* @return Boolean.TRUE if the value was a recognized "true" value,  
198 \* Boolean.FALSE if the value was a recognized "false" value, or  
199 \* the default value if the value was not recognized and the constructor  
200 \* was provided with a default value.  
201 \*  
202 \* @throws Throwable if an error occurs converting to the specified type  
203 \* @since 1.8.0  
204 \*/  
205 @Override  
206 protected <T> T convertToType(final Class<T> type, final Object value) throws Throwable {  
207  
208 if (Boolean.class.equals(type) || Boolean.TYPE.equals(type)) {  
209 // All the values in the trueStrings and falseStrings arrays are  
210 // guaranteed to be lower-case. By converting the input value  
211 // to lowercase too, we can use the efficient String.equals method  
212 // instead of the less-efficient String.equalsIgnoreCase method.  
213 final String stringValue = value.toString().toLowerCase();  
214  
215 for (String trueString : trueStrings) {  
216 if (trueString.equals(stringValue)) {  
217 return type.cast(Boolean.TRUE);  
218 }  
219 }  
220  
221 for (String falseString : falseStrings) {  
222 if (falseString.equals(stringValue)) {  
223 return type.cast(Boolean.FALSE);  
224 }  
225 }  
226 }  
227  
228 throw conversionException(type, value);  
229 }  
230  
231 /\*\*  
232 \* This method creates a copy of the provided array, and ensures that  
233 \* all the strings in the newly created array contain only lower-case  
234 \* letters.  
235 \* <p>  
236 \* Using this method to copy string arrays means that changes to the  
237 \* src array do not modify the dst array.  
238 \*/  
239 private static String[] copyStrings(final String[] src) {  
240 final String[] dst = new String[src.length];  
241 for(int i=0; i<src.length; ++i) {  
242 dst[i] = src[i].toLowerCase();  
243 }  
244 return dst;  
245 }  
246}