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}  
021 \* implementation that converts an incoming  
022 \* object into a <code>java.lang.String</code> object.  
023 \* <p>  
024 \* Note that ConvertUtils really is designed to do string->object conversions,  
025 \* and offers very little support for object->string conversions. The  
026 \* ConvertUtils/ConvertUtilsBean methods only select a converter to apply  
027 \* based upon the target type being converted to, and generally assume that  
028 \* the input is a string (by calling its toString method if needed).  
029 \* <p>  
030 \* This class is therefore just a dummy converter that converts its input  
031 \* into a string by calling the input object's toString method and returning  
032 \* that value.  
033 \* <p>  
034 \* It is possible to replace this converter with something that has a big  
035 \* if/else statement that selects behaviour based on the real type of the  
036 \* object being converted (or possibly has a map of converters, and looks  
037 \* them up based on the class of the input object). However this is not part  
038 \* of the existing ConvertUtils framework.  
039 \*  
040 \*  
041 \* @version $Id$  
042 \* @since 1.3  
043 \*/  
044public final class StringConverter extends AbstractConverter {  
045  
046  
047 /\*\*  
048 \* Construct a <b>java.lang.String</b> <i>Converter</i> that throws  
049 \* a <code>ConversionException</code> if an error occurs.  
050 \*/  
051 public StringConverter() {  
052 super();  
053 }  
054  
055 /\*\*  
056 \* Construct a <b>java.lang.String</b> <i>Converter</i> that returns  
057 \* a default value if an error occurs.  
058 \*  
059 \* @param defaultValue The default value to be returned  
060 \* if the value to be converted is missing or an error  
061 \* occurs converting the value.  
062 \*/  
063 public StringConverter(final Object defaultValue) {  
064 super(defaultValue);  
065 }  
066  
067 /\*\*  
068 \* Return the default type this <code>Converter</code> handles.  
069 \*  
070 \* @return The default type this <code>Converter</code> handles.  
071 \* @since 1.8.0  
072 \*/  
073 @Override  
074 protected Class<?> getDefaultType() {  
075 return String.class;  
076 }  
077  
078 /\*\*  
079 \* Convert the specified input object into an output object of the  
080 \* specified type.  
081 \*  
082 \* @param <T> Target type of the conversion.  
083 \* @param type Data type to which this value should be converted.  
084 \* @param value The input value to be converted.  
085 \* @return The converted value.  
086 \* @throws Throwable if an error occurs converting to the specified type  
087 \* @since 1.8.0  
088 \*/  
089 @Override  
090 protected <T> T convertToType(final Class<T> type, final Object value) throws Throwable {  
091 // We have to support Object, too, because this class is sometimes  
092 // used for a standard to Object conversion  
093 if (String.class.equals(type) || Object.class.equals(type)) {  
094 return type.cast(value.toString());  
095 }  
096 throw conversionException(type, value);  
097 }  
098  
099  
100}