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  
018package org.apache.commons.beanutils;  
019  
020import java.io.Serializable;  
021import java.lang.reflect.InvocationTargetException;  
022import java.util.Comparator;  
023  
024import org.apache.commons.collections.comparators.ComparableComparator;  
025  
026/\*\*  
027 \* <p>  
028 \* This comparator compares two beans by the specified bean property.  
029 \* It is also possible to compare beans based on nested, indexed,  
030 \* combined, mapped bean properties. Please see the {@link PropertyUtilsBean}  
031 \* documentation for all property name possibilities.  
032 \*  
033 \* </p><p>  
034 \* <strong>Note:</strong> The BeanComparator passes the values of the specified  
035 \* bean property to a ComparableComparator, if no comparator is  
036 \* specified in the constructor. If you are comparing two beans based  
037 \* on a property that could contain "null" values, a suitable <code>Comparator</code>  
038 \* or <code>ComparatorChain</code> should be supplied in the constructor.  
039 \* Note that the passed in {@code Comparator} must be able to handle the  
040 \* passed in objects. Because the type of the property to be compared is not  
041 \* known at compile time no type checks can be performed by the compiler.  
042 \* Thus {@code ClassCastException} exceptions can be thrown if unexpected  
043 \* property values occur.  
044 \* </p>  
045 \*  
046 \* @param <T> the type of beans to be compared by this {@code Comparator}  
047 \* @version $Id$  
048 \*/  
049public class BeanComparator<T> implements Comparator<T>, Serializable {  
050  
051 private String property;  
052 private final Comparator<?> comparator;  
053  
054 /\*\*  
055 \* <p>Constructs a Bean Comparator without a property set.  
056 \* </p><p>  
057 \* <strong>Note</strong> that this is intended to be used  
058 \* only in bean-centric environments.  
059 \* </p><p>  
060 \* Until {@link #setProperty} is called with a non-null value.  
061 \* this comparator will compare the Objects only.  
062 \* </p>  
063 \*/  
064 public BeanComparator() {  
065 this( null );  
066 }  
067  
068 /\*\*  
069 \* <p>Constructs a property-based comparator for beans.  
070 \* This compares two beans by the property  
071 \* specified in the property parameter. This constructor creates  
072 \* a <code>BeanComparator</code> that uses a <code>ComparableComparator</code>  
073 \* to compare the property values.  
074 \* </p>  
075 \*  
076 \* <p>Passing "null" to this constructor will cause the BeanComparator  
077 \* to compare objects based on natural order, that is  
078 \* <code>java.lang.Comparable</code>.  
079 \* </p>  
080 \*  
081 \* @param property String Name of a bean property, which may contain the  
082 \* name of a simple, nested, indexed, mapped, or combined  
083 \* property. See {@link PropertyUtilsBean} for property query language syntax.  
084 \* If the property passed in is null then the actual objects will be compared  
085 \*/  
086 public BeanComparator( final String property ) {  
087 this( property, ComparableComparator.getInstance() );  
088 }  
089  
090 /\*\*  
091 \* Constructs a property-based comparator for beans.  
092 \* This constructor creates  
093 \* a BeanComparator that uses the supplied Comparator to compare  
094 \* the property values.  
095 \*  
096 \* @param property Name of a bean property, can contain the name  
097 \* of a simple, nested, indexed, mapped, or combined  
098 \* property. See {@link PropertyUtilsBean} for property query language  
099 \* syntax.  
100 \* @param comparator BeanComparator will pass the values of the  
101 \* specified bean property to this Comparator.  
102 \* If your bean property is not a comparable or  
103 \* contains null values, a suitable comparator  
104 \* may be supplied in this constructor.  
105 \*/  
106 public BeanComparator( final String property, final Comparator<?> comparator ) {  
107 setProperty( property );  
108 if (comparator != null) {  
109 this.comparator = comparator;  
110 } else {  
111 this.comparator = ComparableComparator.getInstance();  
112 }  
113 }  
114  
115 /\*\*  
116 \* Sets the method to be called to compare two JavaBeans  
117 \*  
118 \* @param property String method name to call to compare  
119 \* If the property passed in is null then the actual objects will be compared  
120 \*/  
121 public void setProperty( final String property ) {  
122 this.property = property;  
123 }  
124  
125  
126 /\*\*  
127 \* Gets the property attribute of the BeanComparator  
128 \*  
129 \* @return String method name to call to compare.  
130 \* A null value indicates that the actual objects will be compared  
131 \*/  
132 public String getProperty() {  
133 return property;  
134 }  
135  
136  
137 /\*\*  
138 \* Gets the Comparator being used to compare beans.  
139 \*  
140 \* @return the Comparator being used to compare beans  
141 \*/  
142 public Comparator<?> getComparator() {  
143 return comparator;  
144 }  
145  
146  
147 /\*\*  
148 \* Compare two JavaBeans by their shared property.  
149 \* If {@link #getProperty} is null then the actual objects will be compared.  
150 \*  
151 \* @param o1 Object The first bean to get data from to compare against  
152 \* @param o2 Object The second bean to get data from to compare  
153 \* @return int negative or positive based on order  
154 \*/  
155 public int compare( final T o1, final T o2 ) {  
156  
157 if ( property == null ) {  
158 // compare the actual objects  
159 return internalCompare( o1, o2 );  
160 }  
161  
162 try {  
163 final Object value1 = PropertyUtils.getProperty( o1, property );  
164 final Object value2 = PropertyUtils.getProperty( o2, property );  
165 return internalCompare( value1, value2 );  
166 }  
167 catch ( final IllegalAccessException iae ) {  
168 throw new RuntimeException( "IllegalAccessException: " + iae.toString() );  
169 }  
170 catch ( final InvocationTargetException ite ) {  
171 throw new RuntimeException( "InvocationTargetException: " + ite.toString() );  
172 }  
173 catch ( final NoSuchMethodException nsme ) {  
174 throw new RuntimeException( "NoSuchMethodException: " + nsme.toString() );  
175 }  
176 }  
177  
178 /\*\*  
179 \* Two <code>BeanComparator</code>'s are equals if and only if  
180 \* the wrapped comparators and the property names to be compared  
181 \* are equal.  
182 \* @param o Comparator to compare to  
183 \* @return whether the the comparators are equal or not  
184 \*/  
185 @Override  
186 public boolean equals(final Object o) {  
187 if (this == o) {  
188 return true;  
189 }  
190 if (!(o instanceof BeanComparator)) {  
191 return false;  
192 }  
193  
194 final BeanComparator<?> beanComparator = (BeanComparator<?>) o;  
195  
196 if (!comparator.equals(beanComparator.comparator)) {  
197 return false;  
198 }  
199 if (property != null)  
200 {  
201 if (!property.equals(beanComparator.property)) {  
202 return false;  
203 }  
204 }  
205 else  
206 {  
207 return (beanComparator.property == null);  
208 }  
209  
210 return true;  
211 }  
212  
213 /\*\*  
214 \* Hashcode compatible with equals.  
215 \* @return the hash code for this comparator  
216 \*/  
217 @Override  
218 public int hashCode() {  
219 int result;  
220 result = comparator.hashCode();  
221 return result;  
222 }  
223  
224 /\*\*  
225 \* Compares the given values using the internal {@code Comparator}.  
226 \* <em>Note</em>: This comparison cannot be performed in a type-safe way; so  
227 \* {@code ClassCastException} exceptions may be thrown.  
228 \*  
229 \* @param val1 the first value to be compared  
230 \* @param val2 the second value to be compared  
231 \* @return the result of the comparison  
232 \*/  
233 private int internalCompare(final Object val1, final Object val2) {  
234 @SuppressWarnings("rawtypes")  
235 final  
236 // to make the compiler happy  
237 Comparator c = comparator;  
238 return c.compare(val1, val2);  
239 }  
240}