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.collections4.comparators;  
018  
019import java.io.Serializable;  
020import java.util.Comparator;  
021  
022import org.apache.commons.collections4.ComparatorUtils;  
023import org.apache.commons.collections4.Transformer;  
024  
025/\*\*  
026 \* Decorates another Comparator with transformation behavior. That is, the  
027 \* return value from the transform operation will be passed to the decorated  
028 \* {@link Comparator#compare(Object,Object) compare} method.  
029 \* <p>  
030 \* This class is Serializable from Commons Collections 4.0.  
031 \* </p>  
032 \*  
033 \* @param <I> the input type to the transformer  
034 \* @param <O> the output type from the transformer  
035 \*  
036 \* @since 2.1  
037 \*  
038 \* @see org.apache.commons.collections4.Transformer  
039 \* @see org.apache.commons.collections4.comparators.ComparableComparator  
040 \*/  
041public class TransformingComparator<I, O> implements Comparator<I>, Serializable {  
042  
043 /\*\* Serialization version from Collections 4.0. \*/  
044 private static final long serialVersionUID = 3456940356043606220L;  
045  
046 /\*\* The decorated comparator. \*/  
047 private final Comparator<O> decorated;  
048 /\*\* The transformer being used. \*/  
049 private final Transformer<? super I, ? extends O> transformer;  
050  
051 //-----------------------------------------------------------------------  
052 /\*\*  
053 \* Constructs an instance with the given Transformer and a  
054 \* {@link ComparableComparator ComparableComparator}.  
055 \*  
056 \* @param transformer what will transform the arguments to <code>compare</code>  
057 \*/  
058 public TransformingComparator(final Transformer<? super I, ? extends O> transformer) {  
059 this(transformer, ComparatorUtils.NATURAL\_COMPARATOR);  
060 }  
061  
062 /\*\*  
063 \* Constructs an instance with the given Transformer and Comparator.  
064 \*  
065 \* @param transformer what will transform the arguments to <code>compare</code>  
066 \* @param decorated the decorated Comparator  
067 \*/  
068 public TransformingComparator(final Transformer<? super I, ? extends O> transformer,  
069 final Comparator<O> decorated) {  
070 this.decorated = decorated;  
071 this.transformer = transformer;  
072 }  
073  
074 //-----------------------------------------------------------------------  
075 /\*\*  
076 \* Returns the result of comparing the values from the transform operation.  
077 \*  
078 \* @param obj1 the first object to transform then compare  
079 \* @param obj2 the second object to transform then compare  
080 \* @return negative if obj1 is less, positive if greater, zero if equal  
081 \*/  
082 @Override  
083 public int compare(final I obj1, final I obj2) {  
084 final O value1 = this.transformer.transform(obj1);  
085 final O value2 = this.transformer.transform(obj2);  
086 return this.decorated.compare(value1, value2);  
087 }  
088  
089 //-----------------------------------------------------------------------  
090 /\*\*  
091 \* Implement a hash code for this comparator that is consistent with  
092 \* {@link #equals(Object) equals}.  
093 \*  
094 \* @return a hash code for this comparator.  
095 \*/  
096 @Override  
097 public int hashCode() {  
098 int total = 17;  
099 total = total\*37 + (decorated == null ? 0 : decorated.hashCode());  
100 total = total\*37 + (transformer == null ? 0 : transformer.hashCode());  
101 return total;  
102 }  
103  
104 /\*\*  
105 \* Returns <code>true</code> iff <i>that</i> Object is  
106 \* is a {@link Comparator} whose ordering is known to be  
107 \* equivalent to mine.  
108 \* <p>  
109 \* This implementation returns <code>true</code>  
110 \* iff <code><i>that</i></code> is a {@link TransformingComparator}  
111 \* whose attributes are equal to mine.  
112 \*  
113 \* @param object the object to compare to  
114 \* @return true if equal  
115 \*/  
116 @Override  
117 public boolean equals(final Object object) {  
118 if (this == object) {  
119 return true;  
120 }  
121 if (null == object) {  
122 return false;  
123 }  
124 if (object.getClass().equals(this.getClass())) {  
125 final TransformingComparator<?, ?> comp = (TransformingComparator<?, ?>) object;  
126 return (null == decorated ? null == comp.decorated : decorated.equals(comp.decorated)) &&  
127 (null == transformer ? null == comp.transformer : transformer.equals(comp.transformer));  
128 }  
129 return false;  
130 }  
131  
132}  
133