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;  
023  
024/\*\*  
025 \* Reverses the order of another comparator by reversing the arguments  
026 \* to its {@link #compare(Object, Object) compare} method.  
027 \*  
028 \* @param <E> the type of objects compared by this comparator  
029 \*  
030 \* @since 2.0  
031 \* @see java.util.Collections#reverseOrder()  
032 \*/  
033public class ReverseComparator<E> implements Comparator<E>, Serializable {  
034  
035 /\*\* Serialization version from Collections 2.0. \*/  
036 private static final long serialVersionUID = 2858887242028539265L;  
037  
038 /\*\* The comparator being decorated. \*/  
039 private final Comparator<? super E> comparator;  
040  
041 //-----------------------------------------------------------------------  
042 /\*\*  
043 \* Creates a comparator that compares objects based on the inverse of their  
044 \* natural ordering. Using this Constructor will create a ReverseComparator  
045 \* that is functionally identical to the Comparator returned by  
046 \* java.util.Collections.<b>reverseOrder()</b>.  
047 \*  
048 \* @see java.util.Collections#reverseOrder()  
049 \*/  
050 public ReverseComparator() {  
051 this(null);  
052 }  
053  
054 /\*\*  
055 \* Creates a comparator that inverts the comparison  
056 \* of the given comparator. If you pass in <code>null</code>,  
057 \* the ReverseComparator defaults to reversing the  
058 \* natural order, as per {@link java.util.Collections#reverseOrder()}.  
059 \*  
060 \* @param comparator Comparator to reverse  
061 \*/  
062 public ReverseComparator(final Comparator<? super E> comparator) {  
063 this.comparator = comparator == null ? ComparatorUtils.NATURAL\_COMPARATOR : comparator;  
064 }  
065  
066 //-----------------------------------------------------------------------  
067 /\*\*  
068 \* Compares two objects in reverse order.  
069 \*  
070 \* @param obj1 the first object to compare  
071 \* @param obj2 the second object to compare  
072 \* @return negative if obj1 is less, positive if greater, zero if equal  
073 \*/  
074 @Override  
075 public int compare(final E obj1, final E obj2) {  
076 return comparator.compare(obj2, obj1);  
077 }  
078  
079 //-----------------------------------------------------------------------  
080 /\*\*  
081 \* Implement a hash code for this comparator that is consistent with  
082 \* {@link #equals(Object) equals}.  
083 \*  
084 \* @return a suitable hash code  
085 \* @since 3.0  
086 \*/  
087 @Override  
088 public int hashCode() {  
089 return "ReverseComparator".hashCode() ^ comparator.hashCode();  
090 }  
091  
092 /\*\*  
093 \* Returns <code>true</code> iff <i>that</i> Object is  
094 \* is a {@link Comparator} whose ordering is known to be  
095 \* equivalent to mine.  
096 \* <p>  
097 \* This implementation returns <code>true</code>  
098 \* iff <code><i>object</i>.{@link Object#getClass() getClass()}</code>  
099 \* equals <code>this.getClass()</code>, and the underlying  
100 \* comparators are equal.  
101 \* Subclasses may want to override this behavior to remain consistent  
102 \* with the {@link Comparator#equals(Object) equals} contract.  
103 \*  
104 \* @param object the object to compare to  
105 \* @return true if equal  
106 \* @since 3.0  
107 \*/  
108 @Override  
109 public boolean equals(final Object object) {  
110 if (this == object) {  
111 return true;  
112 }  
113 if (null == object) {  
114 return false;  
115 }  
116 if (object.getClass().equals(this.getClass())) {  
117 final ReverseComparator<?> thatrc = (ReverseComparator<?>) object;  
118 return comparator.equals(thatrc.comparator);  
119 }  
120 return false;  
121 }  
122  
123}