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 \* A Comparator that will compare nulls to be either lower or higher than  
026 \* other objects.  
027 \*  
028 \* @param <E> the type of objects compared by this comparator  
029 \* @since 2.0  
030 \*/  
031public class NullComparator<E> implements Comparator<E>, Serializable {  
032  
033 /\*\* Serialization version. \*/  
034 private static final long serialVersionUID = -5820772575483504339L;  
035  
036 /\*\*  
037 \* The comparator to use when comparing two non-<code>null</code> objects.  
038 \*\*/  
039 private final Comparator<? super E> nonNullComparator;  
040  
041 /\*\*  
042 \* Specifies whether a <code>null</code> are compared as higher than  
043 \* non-<code>null</code> objects.  
044 \*\*/  
045 private final boolean nullsAreHigh;  
046  
047 //-----------------------------------------------------------------------  
048 /\*\*  
049 \* Construct an instance that sorts <code>null</code> higher than any  
050 \* non-<code>null</code> object it is compared with. When comparing two  
051 \* non-<code>null</code> objects, the {@link ComparableComparator} is  
052 \* used.  
053 \*\*/  
054 public NullComparator() {  
055 this(ComparatorUtils.NATURAL\_COMPARATOR, true);  
056 }  
057  
058 /\*\*  
059 \* Construct an instance that sorts <code>null</code> higher than any  
060 \* non-<code>null</code> object it is compared with. When comparing two  
061 \* non-<code>null</code> objects, the specified {@link Comparator} is  
062 \* used.  
063 \*  
064 \* @param nonNullComparator the comparator to use when comparing two  
065 \* non-<code>null</code> objects. This argument cannot be  
066 \* <code>null</code>  
067 \*  
068 \* @throws NullPointerException if <code>nonNullComparator</code> is  
069 \* <code>null</code>  
070 \*\*/  
071 public NullComparator(final Comparator<? super E> nonNullComparator) {  
072 this(nonNullComparator, true);  
073 }  
074  
075 /\*\*  
076 \* Construct an instance that sorts <code>null</code> higher or lower than  
077 \* any non-<code>null</code> object it is compared with. When comparing  
078 \* two non-<code>null</code> objects, the {@link ComparableComparator} is  
079 \* used.  
080 \*  
081 \* @param nullsAreHigh a <code>true</code> value indicates that  
082 \* <code>null</code> should be compared as higher than a  
083 \* non-<code>null</code> object. A <code>false</code> value indicates  
084 \* that <code>null</code> should be compared as lower than a  
085 \* non-<code>null</code> object.  
086 \*\*/  
087 public NullComparator(final boolean nullsAreHigh) {  
088 this(ComparatorUtils.NATURAL\_COMPARATOR, nullsAreHigh);  
089 }  
090  
091 /\*\*  
092 \* Construct an instance that sorts <code>null</code> higher or lower than  
093 \* any non-<code>null</code> object it is compared with. When comparing  
094 \* two non-<code>null</code> objects, the specified {@link Comparator} is  
095 \* used.  
096 \*  
097 \* @param nonNullComparator the comparator to use when comparing two  
098 \* non-<code>null</code> objects. This argument cannot be  
099 \* <code>null</code>  
100 \*  
101 \* @param nullsAreHigh a <code>true</code> value indicates that  
102 \* <code>null</code> should be compared as higher than a  
103 \* non-<code>null</code> object. A <code>false</code> value indicates  
104 \* that <code>null</code> should be compared as lower than a  
105 \* non-<code>null</code> object.  
106 \*  
107 \* @throws NullPointerException if <code>nonNullComparator</code> is  
108 \* <code>null</code>  
109 \*\*/  
110 public NullComparator(final Comparator<? super E> nonNullComparator, final boolean nullsAreHigh) {  
111 this.nonNullComparator = nonNullComparator;  
112 this.nullsAreHigh = nullsAreHigh;  
113  
114 if (nonNullComparator == null) {  
115 throw new NullPointerException("null nonNullComparator");  
116 }  
117 }  
118  
119 //-----------------------------------------------------------------------  
120 /\*\*  
121 \* Perform a comparison between two objects. If both objects are  
122 \* <code>null</code>, a <code>0</code> value is returned. If one object  
123 \* is <code>null</code> and the other is not, the result is determined on  
124 \* whether the Comparator was constructed to have nulls as higher or lower  
125 \* than other objects. If neither object is <code>null</code>, an  
126 \* underlying comparator specified in the constructor (or the default) is  
127 \* used to compare the non-<code>null</code> objects.  
128 \*  
129 \* @param o1 the first object to compare  
130 \* @param o2 the object to compare it to.  
131 \* @return <code>-1</code> if <code>o1</code> is "lower" than (less than,  
132 \* before, etc.) <code>o2</code>; <code>1</code> if <code>o1</code> is  
133 \* "higher" than (greater than, after, etc.) <code>o2</code>; or  
134 \* <code>0</code> if <code>o1</code> and <code>o2</code> are equal.  
135 \*\*/  
136 @Override  
137 public int compare(final E o1, final E o2) {  
138 if(o1 == o2) { return 0; }  
139 if(o1 == null) { return this.nullsAreHigh ? 1 : -1; }  
140 if(o2 == null) { return this.nullsAreHigh ? -1 : 1; }  
141 return this.nonNullComparator.compare(o1, o2);  
142 }  
143  
144 //-----------------------------------------------------------------------  
145 /\*\*  
146 \* Implement a hash code for this comparator that is consistent with  
147 \* {@link #equals(Object)}.  
148 \*  
149 \* @return a hash code for this comparator.  
150 \*\*/  
151 @Override  
152 public int hashCode() {  
153 return (nullsAreHigh ? -1 : 1) \* nonNullComparator.hashCode();  
154 }  
155  
156 /\*\*  
157 \* Determines whether the specified object represents a comparator that is  
158 \* equal to this comparator.  
159 \*  
160 \* @param obj the object to compare this comparator with.  
161 \*  
162 \* @return <code>true</code> if the specified object is a NullComparator  
163 \* with equivalent <code>null</code> comparison behavior  
164 \* (i.e. <code>null</code> high or low) and with equivalent underlying  
165 \* non-<code>null</code> object comparators.  
166 \*\*/  
167 @Override  
168 public boolean equals(final Object obj) {  
169 if(obj == null) { return false; }  
170 if(obj == this) { return true; }  
171 if(!obj.getClass().equals(this.getClass())) { return false; }  
172  
173 final NullComparator<?> other = (NullComparator<?>) obj;  
174  
175 return this.nullsAreHigh == other.nullsAreHigh &&  
176 this.nonNullComparator.equals(other.nonNullComparator);  
177 }  
178  
179}