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  
022/\*\*  
023 \* A {@link Comparator} for {@link Boolean} objects that can sort either  
024 \* true or false first.  
025 \*  
026 \* @see #getTrueFirstComparator()  
027 \* @see #getFalseFirstComparator()  
028 \* @see #booleanComparator(boolean)  
029 \*  
030 \* @since 3.0  
031 \*/  
032public final class BooleanComparator implements Comparator<Boolean>, Serializable {  
033  
034 /\*\* Serialization version. \*/  
035 private static final long serialVersionUID = 1830042991606340609L;  
036  
037 /\*\* Constant "true first" reference. \*/  
038 private static final BooleanComparator TRUE\_FIRST = new BooleanComparator(true);  
039  
040 /\*\* Constant "false first" reference. \*/  
041 private static final BooleanComparator FALSE\_FIRST = new BooleanComparator(false);  
042  
043 /\*\* <code>true</code> iff <code>true</code> values sort before <code>false</code> values. \*/  
044 private boolean trueFirst = false;  
045  
046 //-----------------------------------------------------------------------  
047 /\*\*  
048 \* Returns a BooleanComparator instance that sorts  
049 \* <code>true</code> values before <code>false</code> values.  
050 \* <p>  
051 \* Clients are encouraged to use the value returned from  
052 \* this method instead of constructing a new instance  
053 \* to reduce allocation and garbage collection overhead when  
054 \* multiple BooleanComparators may be used in the same  
055 \* virtual machine.  
056 \* </p>  
057 \*  
058 \* @return the true first singleton BooleanComparator  
059 \*/  
060 public static BooleanComparator getTrueFirstComparator() {  
061 return TRUE\_FIRST;  
062 }  
063  
064 /\*\*  
065 \* Returns a BooleanComparator instance that sorts  
066 \* <code>false</code> values before <code>true</code> values.  
067 \* <p>  
068 \* Clients are encouraged to use the value returned from  
069 \* this method instead of constructing a new instance  
070 \* to reduce allocation and garbage collection overhead when  
071 \* multiple BooleanComparators may be used in the same  
072 \* virtual machine.  
073 \* </p>  
074 \*  
075 \* @return the false first singleton BooleanComparator  
076 \*/  
077 public static BooleanComparator getFalseFirstComparator() {  
078 return FALSE\_FIRST;  
079 }  
080  
081 /\*\*  
082 \* Returns a BooleanComparator instance that sorts  
083 \* <code><i>trueFirst</i></code> values before  
084 \* <code>!<i>trueFirst</i></code> values.  
085 \* <p>  
086 \* Clients are encouraged to use the value returned from  
087 \* this method instead of constructing a new instance  
088 \* to reduce allocation and garbage collection overhead when  
089 \* multiple BooleanComparators may be used in the same  
090 \* virtual machine.  
091 \* </p>  
092 \*  
093 \* @param trueFirst when <code>true</code>, sort  
094 \* <code>true</code> <code>Boolean</code>s before <code>false</code>  
095 \* @return a singleton BooleanComparator instance  
096 \* @since 4.0  
097 \*/  
098 public static BooleanComparator booleanComparator(final boolean trueFirst) {  
099 return trueFirst ? TRUE\_FIRST : FALSE\_FIRST;  
100 }  
101  
102 //-----------------------------------------------------------------------  
103 /\*\*  
104 \* Creates a <code>BooleanComparator</code> that sorts  
105 \* <code>false</code> values before <code>true</code> values.  
106 \* <p>  
107 \* Equivalent to {@link #BooleanComparator(boolean) BooleanComparator(false)}.  
108 \* <p>  
109 \* Please use the static factory instead whenever possible.  
110 \*/  
111 public BooleanComparator() {  
112 this(false);  
113 }  
114  
115 /\*\*  
116 \* Creates a <code>BooleanComparator</code> that sorts  
117 \* <code><i>trueFirst</i></code> values before  
118 \* <code>!<i>trueFirst</i></code> values.  
119 \* <p>  
120 \* Please use the static factories instead whenever possible.  
121 \*  
122 \* @param trueFirst when <code>true</code>, sort  
123 \* <code>true</code> boolean values before <code>false</code>  
124 \*/  
125 public BooleanComparator(final boolean trueFirst) {  
126 this.trueFirst = trueFirst;  
127 }  
128  
129 //-----------------------------------------------------------------------  
130 /\*\*  
131 \* Compares two non-<code>null</code> <code>Boolean</code> objects  
132 \* according to the value of {@link #sortsTrueFirst()}.  
133 \*  
134 \* @param b1 the first boolean to compare  
135 \* @param b2 the second boolean to compare  
136 \* @return negative if obj1 is less, positive if greater, zero if equal  
137 \* @throws NullPointerException when either argument <code>null</code>  
138 \*/  
139 @Override  
140 public int compare(final Boolean b1, final Boolean b2) {  
141 final boolean v1 = b1.booleanValue();  
142 final boolean v2 = b2.booleanValue();  
143  
144 return (v1 ^ v2) ? ( (v1 ^ trueFirst) ? 1 : -1 ) : 0;  
145 }  
146  
147 //-----------------------------------------------------------------------  
148 /\*\*  
149 \* Implement a hash code for this comparator that is consistent with  
150 \* {@link #equals(Object) equals}.  
151 \*  
152 \* @return a hash code for this comparator.  
153 \*/  
154 @Override  
155 public int hashCode() {  
156 final int hash = "BooleanComparator".hashCode();  
157 return trueFirst ? -1 \* hash : hash;  
158 }  
159  
160 /\*\*  
161 \* Returns <code>true</code> iff <i>that</i> Object is  
162 \* is a {@link Comparator} whose ordering is known to be  
163 \* equivalent to mine.  
164 \* <p>  
165 \* This implementation returns <code>true</code>  
166 \* iff <code><i>that</i></code> is a {@link BooleanComparator}  
167 \* whose value of {@link #sortsTrueFirst()} is equal to mine.  
168 \*  
169 \* @param object the object to compare to  
170 \* @return true if equal  
171 \*/  
172 @Override  
173 public boolean equals(final Object object) {  
174 return (this == object) ||  
175 ((object instanceof BooleanComparator) &&  
176 (this.trueFirst == ((BooleanComparator)object).trueFirst));  
177 }  
178  
179 //-----------------------------------------------------------------------  
180 /\*\*  
181 \* Returns <code>true</code> iff  
182 \* I sort <code>true</code> values before  
183 \* <code>false</code> values. In other words,  
184 \* returns <code>true</code> iff  
185 \* {@link #compare(Boolean,Boolean) compare(Boolean.FALSE,Boolean.TRUE)}  
186 \* returns a positive value.  
187 \*  
188 \* @return the trueFirst flag  
189 \*/  
190 public boolean sortsTrueFirst() {  
191 return trueFirst;  
192 }  
193  
194}