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 Comparator} that compares {@link Comparable Comparable}  
024 \* objects.  
025 \* <p>  
026 \* This Comparator is useful, for example, for enforcing the natural order in  
027 \* custom implementations of {@link java.util.SortedSet SortedSet} and  
028 \* {@link java.util.SortedMap SortedMap}.  
029 \* </p>  
030 \* <p>  
031 \* Note: In the 2.0 and 2.1 releases of Commons Collections, this class would  
032 \* throw a {@link ClassCastException} if either of the arguments to  
033 \* {@link #compare(Object, Object) compare} were <code>null</code>, not  
034 \* {@link Comparable Comparable}, or for which  
035 \* {@link Comparable#compareTo(Object) compareTo} gave inconsistent results.  
036 \* This is no longer the case. See {@link #compare(Object, Object) compare} for  
037 \* details.  
038 \* </p>  
039 \*  
040 \* @param <E> the type of objects compared by this comparator  
041 \*  
042 \* @since 2.0  
043 \* @see java.util.Collections#reverseOrder()  
044 \*/  
045public class ComparableComparator<E extends Comparable<? super E>> implements Comparator<E>, Serializable {  
046  
047 /\*\* Serialization version. \*/  
048 private static final long serialVersionUID=-291439688585137865L;  
049  
050 /\*\* The singleton instance. \*/  
051 @SuppressWarnings("rawtypes")  
052 public static final ComparableComparator INSTANCE = new ComparableComparator();  
053  
054 //-----------------------------------------------------------------------  
055 /\*\*  
056 \* Gets the singleton instance of a ComparableComparator.  
057 \* <p>  
058 \* Developers are encouraged to use the comparator returned from this method  
059 \* instead of constructing a new instance to reduce allocation and GC overhead  
060 \* when multiple comparable comparators may be used in the same VM.  
061 \*  
062 \* @param <E> the element type  
063 \* @return the singleton ComparableComparator  
064 \* @since 4.0  
065 \*/  
066 public static <E extends Comparable<? super E>> ComparableComparator<E> comparableComparator() {  
067 return INSTANCE;  
068 }  
069  
070 //-----------------------------------------------------------------------  
071 /\*\*  
072 \* Constructor whose use should be avoided.  
073 \* <p>  
074 \* Please use the {@link #comparableComparator()} method whenever possible.  
075 \*/  
076 public ComparableComparator() {  
077 super();  
078 }  
079  
080 //-----------------------------------------------------------------------  
081 /\*\*  
082 \* Compare the two {@link Comparable Comparable} arguments.  
083 \* This method is equivalent to:  
084 \* <pre>((Comparable)obj1).compareTo(obj2)</pre>  
085 \*  
086 \* @param obj1 the first object to compare  
087 \* @param obj2 the second object to compare  
088 \* @return negative if obj1 is less, positive if greater, zero if equal  
089 \* @throws NullPointerException if <i>obj1</i> is <code>null</code>,  
090 \* or when <code>((Comparable)obj1).compareTo(obj2)</code> does  
091 \* @throws ClassCastException if <i>obj1</i> is not a <code>Comparable</code>,  
092 \* or when <code>((Comparable)obj1).compareTo(obj2)</code> does  
093 \*/  
094 @Override  
095 public int compare(final E obj1, final E obj2) {  
096 return obj1.compareTo(obj2);  
097 }  
098  
099 //-----------------------------------------------------------------------  
100 /\*\*  
101 \* Implement a hash code for this comparator that is consistent with  
102 \* {@link #equals(Object) equals}.  
103 \*  
104 \* @return a hash code for this comparator.  
105 \* @since 3.0  
106 \*/  
107 @Override  
108 public int hashCode() {  
109 return "ComparableComparator".hashCode();  
110 }  
111  
112 /\*\*  
113 \* Returns {@code true} iff <i>that</i> Object is is a {@link Comparator Comparator}  
114 \* whose ordering is known to be equivalent to mine.  
115 \* <p>  
116 \* This implementation returns {@code true} iff  
117 \* <code><i>object</i>.{@link Object#getClass() getClass()}</code> equals  
118 \* <code>this.getClass()</code>. Subclasses may want to override this behavior to remain  
119 \* consistent with the {@link Comparator#equals(Object)} contract.  
120 \*  
121 \* @param object the object to compare with  
122 \* @return {@code true} if equal  
123 \* @since 3.0  
124 \*/  
125 @Override  
126 public boolean equals(final Object object) {  
127 return this == object ||  
128 null != object && object.getClass().equals(this.getClass());  
129 }  
130  
131}