```java
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.commons.collections4.comparators;

import java.io.Serializable;
import java.util.Comparator;

/**
 * A {@link Comparator Comparator} that compares {@link Comparable Comparable}
 * objects.
 * <p>
 * This Comparator is useful, for example, for enforcing the natural order in
 * custom implementations of {@link java.util.SortedSet SortedSet} and
 * {@link java.util.SortedMap SortedMap}.
 * <p>
 * Note: In the 2.0 and 2.1 releases of Commons Collections, this class would
 * throw a {@link ClassCastException} if either of the arguments to
 * {@link #compare(Object, Object) compare} were <code>null</code>, not
 * {@link Comparable Comparable}, or for which
 * {@link Comparable#compareTo(Object) compareTo} gave inconsistent results.
 * This is no longer the case. See {@link #compare(Object, Object) compare} for
 * details.
 *
 * @param <E> the type of objects compared by this comparator
 *
 * @since 2.0
 * @see java.util.Collections#reverseOrder()
 */
public class ComparableComparator<E extends Comparable<? super E>> implements Comparator<E>, Serializable {

    /** Serialization version. */
    private static final long serialVersionUID=-291439688585137865L;

    /** The singleton instance. */
    @SuppressWarnings("rawtypes")
    public static final ComparableComparator INSTANCE = new ComparableComparator();

    //-----------------------------------------------------------------------
    /**
     * Gets the singleton instance of a ComparableComparator.
     * <p>
     * Developers are encouraged to use the comparator returned from this method
     * instead of constructing a new instance to reduce allocation and GC overhead
     * when multiple comparable comparators may be used in the same VM.
     *
     * @param <E>  the element type
     * @return the singleton ComparableComparator
     * @since 4.0
     */
    @SuppressWarnings("unchecked")
    public static <E extends Comparable<? super E>> ComparableComparator<E> comparableComparator() {
        return INSTANCE;
    }

    //-----------------------------------------------------------------------
    /**
     * Constructor whose use should be avoided.
     * <p>
     * Please use the {@link #comparableComparator()} method whenever possible.
     */
    public ComparableComparator() {
```

```
076             super();
077         }
078 
079         //-----------------------------------------------------------------------
080         /**
081          * Compare the two {@link Comparable Comparable} arguments.
082          * This method is equivalent to:
083          * <pre>((Comparable)obj1).compareTo(obj2)</pre>
084          *
085          * @param obj1  the first object to compare
086          * @param obj2  the second object to compare
087          * @return negative if obj1 is less, positive if greater, zero if equal
088          * @throws NullPointerException if <i>obj1</i> is <code>null</code>,
089          *         or when <code>((Comparable)obj1).compareTo(obj2)</code> does
090          * @throws ClassCastException if <i>obj1</i> is not a <code>Comparable</code>,
091          *         or when <code>((Comparable)obj1).compareTo(obj2)</code> does
092          */
093         @Override
094         public int compare(final E obj1, final E obj2) {
095             return obj1.compareTo(obj2);
096         }
097 
098         //-----------------------------------------------------------------------
099         /**
100          * Implement a hash code for this comparator that is consistent with
101          * {@link #equals(Object) equals}.
102          *
103          * @return a hash code for this comparator.
104          * @since 3.0
105          */
106         @Override
107         public int hashCode() {
108             return "ComparableComparator".hashCode();
109         }
110 
111         /**
112          * Returns {@code true} iff <i>that</i> Object is is a {@link Comparator Comparator}
113          * whose ordering is known to be equivalent to mine.
114          * <p>
115          * This implementation returns {@code true} iff
116          * <code><i>object</i>.{@link Object#getClass() getClass()}</code> equals
117          * <code>this.getClass()</code>. Subclasses may want to override this behavior to remain
118          * consistent with the {@link Comparator#equals(Object)} contract.
119          *
120          * @param object  the object to compare with
121          * @return {@code true} if equal
122          * @since 3.0
123          */
124         @Override
125         public boolean equals(final Object object) {
126             return this == object ||
127                    null != object && object.getClass().equals(this.getClass());
128         }
129 
130     }
```