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.functors;  
018  
019import java.io.Serializable;  
020import java.util.Comparator;  
021  
022import org.apache.commons.collections4.Predicate;  
023  
024/\*\*  
025 \* Predicate that compares the input object with the one stored in the predicate using a comparator.  
026 \* In addition, the comparator result can be evaluated in accordance to a supplied criterion value.  
027 \*  
028 \* <p>In order to demonstrate the use of the predicate, the following variables are declared:</p>  
029 \*  
030 \* <pre>  
031 \* Integer ONE = Integer.valueOf(1);  
032 \* Integer TWO = Integer.valueOf(2);  
033 \*  
034 \* Comparator comparator = new Comparator() {  
035 \*  
036 \* public int compare(Object first, Object second) {  
037 \* return ((Integer) second) - ((Integer) first);  
038 \* }  
039 \*  
040 \* };  
041 \* </pre>  
042 \*  
043 \* <p>Using the declared variables, the <code>ComparatorPredicate</code> can be used used in the  
044 \* following way:</p>  
045 \*  
046 \* <pre>  
047 \* ComparatorPredicate.comparatorPredicate(ONE, comparator).evaluate(TWO);  
048 \* </pre>  
049 \*  
050 \* <p>The input variable <code>TWO</code> in compared to the stored variable <code>ONE</code> using  
051 \* the supplied <code>comparator</code>. This is the default usage of the predicate and will return  
052 \* <code>true</code> if the underlying comparator returns <code>0</code>. In addition to the default  
053 \* usage of the predicate, it is possible to evaluate the comparator's result in several ways. The  
054 \* following {@link Criterion} enumeration values are provided by the predicate:  
055 \* </p>  
056 \*  
057 \* <ul>  
058 \* <li>EQUAL</li>  
059 \* <li>GREATER</li>  
060 \* <li>GREATER\_OR\_EQUAL</li>  
061 \* <li>LESS</li>  
062 \* <li>LESS\_OR\_EQUAL</li>  
063 \* </ul>  
064 \*  
065 \* <p>The following examples demonstrates how these constants can be used in order to manipulate the  
066 \* evaluation of a comparator result.</p>  
067 \*  
068 \* <pre>  
069 \* ComparatorPredicate.comparatorPredicate(ONE, comparator,<b>ComparatorPredicate.Criterion.GREATER</b>).evaluate(TWO);  
070 \* </pre>  
071 \*  
072 \* <p>The input variable TWO is compared to the stored variable ONE using the supplied <code>comparator</code>  
073 \* using the <code>GREATER</code> evaluation criterion constant. This instructs the predicate to  
074 \* return <code>true</code> if the comparator returns a value greater than <code>0</code>.</p>  
075 \*  
076 \* @since 4.0  
077 \*/  
078public class ComparatorPredicate<T> implements Predicate<T>, Serializable {  
079  
080 private static final long serialVersionUID = -1863209236504077399L;  
081  
082 public enum Criterion {  
083 EQUAL, GREATER, LESS, GREATER\_OR\_EQUAL, LESS\_OR\_EQUAL,  
084 }  
085  
086 // Instance variables:  
087  
088 /\*\* The internal object to compare with \*/  
089 private final T object;  
090  
091 /\*\* The comparator to use for comparison \*/  
092 private final Comparator<T> comparator;  
093  
094 /\*\* The comparison evaluation criterion to use \*/  
095 private final Criterion criterion;  
096  
097 /\*\*  
098 \* Factory to create the comparator predicate  
099 \*  
100 \* @param <T> the type that the predicate queries  
101 \* @param object the object to compare to  
102 \* @param comparator the comparator to use for comparison  
103 \* @return the predicate  
104 \* @throws NullPointerException if comparator is null  
105 \*/  
106 public static <T> Predicate<T> comparatorPredicate(final T object, final Comparator<T> comparator) {  
107 return comparatorPredicate(object, comparator, Criterion.EQUAL);  
108 }  
109  
110 /\*\*  
111 \* Factory to create the comparator predicate  
112 \*  
113 \* @param <T> the type that the predicate queries  
114 \* @param object the object to compare to  
115 \* @param comparator the comparator to use for comparison  
116 \* @param criterion the criterion to use to evaluate comparison  
117 \* @return the predicate  
118 \* @throws NullPointerException if comparator or criterion is null  
119 \*/  
120 public static <T> Predicate<T> comparatorPredicate(final T object, final Comparator<T> comparator,  
121 final Criterion criterion) {  
122 if (comparator == null) {  
123 throw new NullPointerException("Comparator must not be null.");  
124 }  
125 if (criterion == null) {  
126 throw new NullPointerException("Criterion must not be null.");  
127 }  
128 return new ComparatorPredicate<>(object, comparator, criterion);  
129 }  
130  
131 /\*\*  
132 \* Constructor that performs no validation.  
133 \* Use <code>comparatorPredicate</code> if you want that.  
134 \*  
135 \* @param object the object to compare to  
136 \* @param comparator the comparator to use for comparison  
137 \* @param criterion the criterion to use to evaluate comparison  
138 \*/  
139 public ComparatorPredicate(final T object, final Comparator<T> comparator, final Criterion criterion) {  
140 super();  
141 this.object = object;  
142 this.comparator = comparator;  
143 this.criterion = criterion;  
144 }  
145  
146 /\*\*  
147 \* Evaluates the predicate. The predicate evaluates to <code>true</code> in the following cases:  
148 \*  
149 \* <ul>  
150 \* <li><code>comparator.compare(object, input) == 0 && criterion == EQUAL</code></li>  
151 \* <li><code>comparator.compare(object, input) < 0 && criterion == LESS</code></li>  
152 \* <li><code>comparator.compare(object, input) > 0 && criterion == GREATER</code></li>  
153 \* <li><code>comparator.compare(object, input) >= 0 && criterion == GREATER\_OR\_EQUAL</code></li>  
154 \* <li><code>comparator.compare(object, input) <= 0 && criterion == LESS\_OR\_EQUAL</code></li>  
155 \* </ul>  
156 \*  
157 \* @see org.apache.commons.collections4.Predicate#evaluate(java.lang.Object)  
158 \* @see java.util.Comparator#compare(java.lang.Object first, java.lang.Object second)  
159 \*  
160 \* @param target the target object to compare to  
161 \* @return {@code true} if the comparison succeeds according to the selected criterion  
162 \* @throws IllegalStateException if the criterion is invalid (really not possible)  
163 \*/  
164 @Override  
165 public boolean evaluate(final T target) {  
166  
167 boolean result = false;  
168 final int comparison = comparator.compare(object, target);  
169 switch (criterion) {  
170 case EQUAL:  
171 result = comparison == 0;  
172 break;  
173 case GREATER:  
174 result = comparison > 0;  
175 break;  
176 case LESS:  
177 result = comparison < 0;  
178 break;  
179 case GREATER\_OR\_EQUAL:  
180 result = comparison >= 0;  
181 break;  
182 case LESS\_OR\_EQUAL:  
183 result = comparison <= 0;  
184 break;  
185 default:  
186 throw new IllegalStateException("The current criterion '" + criterion + "' is invalid.");  
187 }  
188  
189 return result;  
190 }  
191}