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;  
018  
019import java.util.Collection;  
020  
021import org.apache.commons.collections4.functors.AllPredicate;  
022import org.apache.commons.collections4.functors.AndPredicate;  
023import org.apache.commons.collections4.functors.AnyPredicate;  
024import org.apache.commons.collections4.functors.EqualPredicate;  
025import org.apache.commons.collections4.functors.ExceptionPredicate;  
026import org.apache.commons.collections4.functors.FalsePredicate;  
027import org.apache.commons.collections4.functors.IdentityPredicate;  
028import org.apache.commons.collections4.functors.InstanceofPredicate;  
029import org.apache.commons.collections4.functors.InvokerTransformer;  
030import org.apache.commons.collections4.functors.NonePredicate;  
031import org.apache.commons.collections4.functors.NotNullPredicate;  
032import org.apache.commons.collections4.functors.NotPredicate;  
033import org.apache.commons.collections4.functors.NullIsExceptionPredicate;  
034import org.apache.commons.collections4.functors.NullIsFalsePredicate;  
035import org.apache.commons.collections4.functors.NullIsTruePredicate;  
036import org.apache.commons.collections4.functors.NullPredicate;  
037import org.apache.commons.collections4.functors.OnePredicate;  
038import org.apache.commons.collections4.functors.OrPredicate;  
039import org.apache.commons.collections4.functors.TransformedPredicate;  
040import org.apache.commons.collections4.functors.TransformerPredicate;  
041import org.apache.commons.collections4.functors.TruePredicate;  
042import org.apache.commons.collections4.functors.UniquePredicate;  
043  
044/\*\*  
045 \* <code>PredicateUtils</code> provides reference implementations and utilities  
046 \* for the Predicate functor interface. The supplied predicates are:  
047 \* <ul>  
048 \* <li>Invoker - returns the result of a method call on the input object  
049 \* <li>InstanceOf - true if the object is an instanceof a class  
050 \* <li>Equal - true if the object equals() a specified object  
051 \* <li>Identity - true if the object == a specified object  
052 \* <li>Null - true if the object is null  
053 \* <li>NotNull - true if the object is not null  
054 \* <li>Unique - true if the object has not already been evaluated  
055 \* <li>And/All - true if all of the predicates are true  
056 \* <li>Or/Any - true if any of the predicates is true  
057 \* <li>Either/One - true if only one of the predicate is true  
058 \* <li>Neither/None - true if none of the predicates are true  
059 \* <li>Not - true if the predicate is false, and vice versa  
060 \* <li>Transformer - wraps a Transformer as a Predicate  
061 \* <li>True - always return true  
062 \* <li>False - always return false  
063 \* <li>Exception - always throws an exception  
064 \* <li>NullIsException/NullIsFalse/NullIsTrue - check for null input  
065 \* <li>Transformed - transforms the input before calling the predicate  
066 \* </ul>  
067 \* All the supplied predicates are Serializable.  
068 \*  
069 \* @since 3.0  
070 \*/  
071public class PredicateUtils {  
072  
073 /\*\*  
074 \* This class is not normally instantiated.  
075 \*/  
076 private PredicateUtils() {}  
077  
078 // Simple predicates  
079 //-----------------------------------------------------------------------------  
080  
081 /\*\*  
082 \* Gets a Predicate that always throws an exception.  
083 \* This could be useful during testing as a placeholder.  
084 \*  
085 \* @param <T> the type that the predicate queries  
086 \* @return the predicate  
087 \* @see ExceptionPredicate  
088 \*/  
089 public static <T> Predicate<T> exceptionPredicate() {  
090 return ExceptionPredicate.exceptionPredicate();  
091 }  
092  
093 /\*\*  
094 \* Gets a Predicate that always returns true.  
095 \*  
096 \* @param <T> the type that the predicate queries  
097 \* @return the predicate  
098 \* @see TruePredicate  
099 \*/  
100 public static <T> Predicate<T> truePredicate() {  
101 return TruePredicate.truePredicate();  
102 }  
103  
104 /\*\*  
105 \* Gets a Predicate that always returns false.  
106 \*  
107 \* @param <T> the type that the predicate queries  
108 \* @return the predicate  
109 \* @see FalsePredicate  
110 \*/  
111 public static <T> Predicate<T> falsePredicate() {  
112 return FalsePredicate.falsePredicate();  
113 }  
114  
115 /\*\*  
116 \* Gets a Predicate that checks if the input object passed in is null.  
117 \*  
118 \* @param <T> the type that the predicate queries  
119 \* @return the predicate  
120 \* @see NullPredicate  
121 \*/  
122 public static <T> Predicate<T> nullPredicate() {  
123 return NullPredicate.nullPredicate();  
124 }  
125  
126 /\*\*  
127 \* Gets a Predicate that checks if the input object passed in is not null.  
128 \*  
129 \* @param <T> the type that the predicate queries  
130 \* @return the predicate  
131 \* @see NotNullPredicate  
132 \*/  
133 public static <T> Predicate<T> notNullPredicate() {  
134 return NotNullPredicate.notNullPredicate();  
135 }  
136  
137 /\*\*  
138 \* Creates a Predicate that checks if the input object is equal to the  
139 \* specified object using equals().  
140 \*  
141 \* @param <T> the type that the predicate queries  
142 \* @param value the value to compare against  
143 \* @return the predicate  
144 \* @see EqualPredicate  
145 \*/  
146 public static <T> Predicate<T> equalPredicate(final T value) {  
147 return EqualPredicate.equalPredicate(value);  
148 }  
149  
150 /\*\*  
151 \* Creates a Predicate that checks if the input object is equal to the  
152 \* specified object by identity.  
153 \*  
154 \* @param <T> the type that the predicate queries  
155 \* @param value the value to compare against  
156 \* @return the predicate  
157 \* @see IdentityPredicate  
158 \*/  
159 public static <T> Predicate<T> identityPredicate(final T value) {  
160 return IdentityPredicate.identityPredicate(value);  
161 }  
162  
163 /\*\*  
164 \* Creates a Predicate that checks if the object passed in is of  
165 \* a particular type, using instanceof. A <code>null</code> input  
166 \* object will return <code>false</code>.  
167 \*  
168 \* @param type the type to check for, may not be null  
169 \* @return the predicate  
170 \* @throws NullPointerException if the class is null  
171 \* @see InstanceofPredicate  
172 \*/  
173 public static Predicate<Object> instanceofPredicate(final Class<?> type) {  
174 return InstanceofPredicate.instanceOfPredicate(type);  
175 }  
176  
177 /\*\*  
178 \* Creates a Predicate that returns true the first time an object is  
179 \* encountered, and false if the same object is received  
180 \* again. The comparison is by equals(). A <code>null</code> input object  
181 \* is accepted and will return true the first time, and false subsequently  
182 \* as well.  
183 \*  
184 \* @param <T> the type that the predicate queries  
185 \* @return the predicate  
186 \* @see UniquePredicate  
187 \*/  
188 public static <T> Predicate<T> uniquePredicate() {  
189 // must return new instance each time  
190 return UniquePredicate.uniquePredicate();  
191 }  
192  
193 /\*\*  
194 \* Creates a Predicate that invokes a method on the input object.  
195 \* The method must return either a boolean or a non-null Boolean,  
196 \* and have no parameters. If the input object is null, a  
197 \* PredicateException is thrown.  
198 \* <p>  
199 \* For example, <code>PredicateUtils.invokerPredicate("isEmpty");</code>  
200 \* will call the <code>isEmpty</code> method on the input object to  
201 \* determine the predicate result.  
202 \*  
203 \* @param <T> the type that the predicate queries  
204 \* @param methodName the method name to call on the input object, may not be null  
205 \* @return the predicate  
206 \* @throws NullPointerException if the methodName is null.  
207 \* @see InvokerTransformer  
208 \* @see TransformerPredicate  
209 \*/  
210 public static <T> Predicate<T> invokerPredicate(final String methodName) {  
211 // reuse transformer as it has caching - this is lazy really, should have inner class here  
212 return asPredicate(InvokerTransformer.<Object, Boolean>invokerTransformer(methodName));  
213 }  
214  
215 /\*\*  
216 \* Creates a Predicate that invokes a method on the input object.  
217 \* The method must return either a boolean or a non-null Boolean,  
218 \* and have no parameters. If the input object is null, a  
219 \* PredicateException is thrown.  
220 \* <p>  
221 \* For example, <code>PredicateUtils.invokerPredicate("isEmpty");</code>  
222 \* will call the <code>isEmpty</code> method on the input object to  
223 \* determine the predicate result.  
224 \*  
225 \* @param <T> the type that the predicate queries  
226 \* @param methodName the method name to call on the input object, may not be null  
227 \* @param paramTypes the parameter types  
228 \* @param args the arguments  
229 \* @return the predicate  
230 \* @throws NullPointerException if the method name is null  
231 \* @throws IllegalArgumentException if the paramTypes and args don't match  
232 \* @see InvokerTransformer  
233 \* @see TransformerPredicate  
234 \*/  
235 public static <T> Predicate<T> invokerPredicate(final String methodName, final Class<?>[] paramTypes,  
236 final Object[] args) {  
237 // reuse transformer as it has caching - this is lazy really, should have inner class here  
238 return asPredicate(InvokerTransformer.<Object, Boolean>invokerTransformer(methodName, paramTypes, args));  
239 }  
240  
241 // Boolean combinations  
242 //-----------------------------------------------------------------------------  
243  
244 /\*\*  
245 \* Create a new Predicate that returns true only if both of the specified  
246 \* predicates are true.  
247 \*  
248 \* @param <T> the type that the predicate queries  
249 \* @param predicate1 the first predicate, may not be null  
250 \* @param predicate2 the second predicate, may not be null  
251 \* @return the <code>and</code> predicate  
252 \* @throws NullPointerException if either predicate is null  
253 \* @see AndPredicate  
254 \*/  
255 public static <T> Predicate<T> andPredicate(final Predicate<? super T> predicate1,  
256 final Predicate<? super T> predicate2) {  
257 return AndPredicate.andPredicate(predicate1, predicate2);  
258 }  
259  
260 /\*\*  
261 \* Create a new Predicate that returns true only if all of the specified  
262 \* predicates are true.  
263 \* If the array of predicates is empty, then this predicate returns true.  
264 \*  
265 \* @param <T> the type that the predicate queries  
266 \* @param predicates an array of predicates to check, may not be null  
267 \* @return the <code>all</code> predicate  
268 \* @throws NullPointerException if the predicates array is null  
269 \* @throws NullPointerException if any predicate in the array is null  
270 \* @see AllPredicate  
271 \*/  
272 public static <T> Predicate<T> allPredicate(final Predicate<? super T>... predicates) {  
273 return AllPredicate.allPredicate(predicates);  
274 }  
275  
276 /\*\*  
277 \* Create a new Predicate that returns true only if all of the specified  
278 \* predicates are true. The predicates are checked in iterator order.  
279 \* If the collection of predicates is empty, then this predicate returns true.  
280 \*  
281 \* @param <T> the type that the predicate queries  
282 \* @param predicates a collection of predicates to check, may not be null  
283 \* @return the <code>all</code> predicate  
284 \* @throws NullPointerException if the predicates collection is null  
285 \* @throws NullPointerException if any predicate in the collection is null  
286 \* @see AllPredicate  
287 \*/  
288 public static <T> Predicate<T> allPredicate(final Collection<? extends Predicate<? super T>> predicates) {  
289 return AllPredicate.allPredicate(predicates);  
290 }  
291  
292 /\*\*  
293 \* Create a new Predicate that returns true if either of the specified  
294 \* predicates are true.  
295 \*  
296 \* @param <T> the type that the predicate queries  
297 \* @param predicate1 the first predicate, may not be null  
298 \* @param predicate2 the second predicate, may not be null  
299 \* @return the <code>or</code> predicate  
300 \* @throws NullPointerException if either predicate is null  
301 \* @see OrPredicate  
302 \*/  
303 public static <T> Predicate<T> orPredicate(final Predicate<? super T> predicate1,  
304 final Predicate<? super T> predicate2) {  
305 return OrPredicate.orPredicate(predicate1, predicate2);  
306 }  
307  
308 /\*\*  
309 \* Create a new Predicate that returns true if any of the specified  
310 \* predicates are true.  
311 \* If the array of predicates is empty, then this predicate returns false.  
312 \*  
313 \* @param <T> the type that the predicate queries  
314 \* @param predicates an array of predicates to check, may not be null  
315 \* @return the <code>any</code> predicate  
316 \* @throws NullPointerException if the predicates array is null  
317 \* @throws NullPointerException if any predicate in the array is null  
318 \* @see AnyPredicate  
319 \*/  
320 public static <T> Predicate<T> anyPredicate(final Predicate<? super T>... predicates) {  
321 return AnyPredicate.anyPredicate(predicates);  
322 }  
323  
324 /\*\*  
325 \* Create a new Predicate that returns true if any of the specified  
326 \* predicates are true. The predicates are checked in iterator order.  
327 \* If the collection of predicates is empty, then this predicate returns false.  
328 \*  
329 \* @param <T> the type that the predicate queries  
330 \* @param predicates a collection of predicates to check, may not be null  
331 \* @return the <code>any</code> predicate  
332 \* @throws NullPointerException if the predicates collection is null  
333 \* @throws NullPointerException if any predicate in the collection is null  
334 \* @see AnyPredicate  
335 \*/  
336 public static <T> Predicate<T> anyPredicate(final Collection<? extends Predicate<? super T>> predicates) {  
337 return AnyPredicate.anyPredicate(predicates);  
338 }  
339  
340 /\*\*  
341 \* Create a new Predicate that returns true if one, but not both, of the  
342 \* specified predicates are true. XOR  
343 \*  
344 \* @param <T> the type that the predicate queries  
345 \* @param predicate1 the first predicate, may not be null  
346 \* @param predicate2 the second predicate, may not be null  
347 \* @return the <code>either</code> predicate  
348 \* @throws NullPointerException if either predicate is null  
349 \* @see OnePredicate  
350 \*/  
351 public static <T> Predicate<T> eitherPredicate(final Predicate<? super T> predicate1,  
352 final Predicate<? super T> predicate2) {  
353 @SuppressWarnings("unchecked")  
354 final Predicate<T> onePredicate = PredicateUtils.onePredicate(predicate1, predicate2);  
355 return onePredicate;  
356 }  
357  
358 /\*\*  
359 \* Create a new Predicate that returns true if only one of the specified  
360 \* predicates are true.  
361 \* If the array of predicates is empty, then this predicate returns false.  
362 \*  
363 \* @param <T> the type that the predicate queries  
364 \* @param predicates an array of predicates to check, may not be null  
365 \* @return the <code>one</code> predicate  
366 \* @throws NullPointerException if the predicates array is null  
367 \* @throws NullPointerException if any predicate in the array is null  
368 \* @see OnePredicate  
369 \*/  
370 public static <T> Predicate<T> onePredicate(final Predicate<? super T>... predicates) {  
371 return OnePredicate.onePredicate(predicates);  
372 }  
373  
374 /\*\*  
375 \* Create a new Predicate that returns true if only one of the specified  
376 \* predicates are true. The predicates are checked in iterator order.  
377 \* If the collection of predicates is empty, then this predicate returns false.  
378 \*  
379 \* @param <T> the type that the predicate queries  
380 \* @param predicates a collection of predicates to check, may not be null  
381 \* @return the <code>one</code> predicate  
382 \* @throws NullPointerException if the predicates collection is null  
383 \* @throws NullPointerException if any predicate in the collection is null  
384 \* @see OnePredicate  
385 \*/  
386 public static <T> Predicate<T> onePredicate(final Collection<? extends Predicate<? super T>> predicates) {  
387 return OnePredicate.onePredicate(predicates);  
388 }  
389  
390 /\*\*  
391 \* Create a new Predicate that returns true if neither of the specified  
392 \* predicates are true.  
393 \*  
394 \* @param <T> the type that the predicate queries  
395 \* @param predicate1 the first predicate, may not be null  
396 \* @param predicate2 the second predicate, may not be null  
397 \* @return the <code>neither</code> predicate  
398 \* @throws NullPointerException if either predicate is null  
399 \* @see NonePredicate  
400 \*/  
401 public static <T> Predicate<T> neitherPredicate(final Predicate<? super T> predicate1,  
402 final Predicate<? super T> predicate2) {  
403 @SuppressWarnings("unchecked")  
404 final Predicate<T> nonePredicate = PredicateUtils.nonePredicate(predicate1, predicate2);  
405 return nonePredicate;  
406 }  
407  
408 /\*\*  
409 \* Create a new Predicate that returns true if none of the specified  
410 \* predicates are true.  
411 \* If the array of predicates is empty, then this predicate returns true.  
412 \*  
413 \* @param <T> the type that the predicate queries  
414 \* @param predicates an array of predicates to check, may not be null  
415 \* @return the <code>none</code> predicate  
416 \* @throws NullPointerException if the predicates array is null  
417 \* @throws NullPointerException if any predicate in the array is null  
418 \* @see NonePredicate  
419 \*/  
420 public static <T> Predicate<T> nonePredicate(final Predicate<? super T>... predicates) {  
421 return NonePredicate.nonePredicate(predicates);  
422 }  
423  
424 /\*\*  
425 \* Create a new Predicate that returns true if none of the specified  
426 \* predicates are true. The predicates are checked in iterator order.  
427 \* If the collection of predicates is empty, then this predicate returns true.  
428 \*  
429 \* @param <T> the type that the predicate queries  
430 \* @param predicates a collection of predicates to check, may not be null  
431 \* @return the <code>none</code> predicate  
432 \* @throws NullPointerException if the predicates collection is null  
433 \* @throws NullPointerException if any predicate in the collection is null  
434 \* @see NonePredicate  
435 \*/  
436 public static <T> Predicate<T> nonePredicate(final Collection<? extends Predicate<? super T>> predicates) {  
437 return NonePredicate.nonePredicate(predicates);  
438 }  
439  
440 /\*\*  
441 \* Create a new Predicate that returns true if the specified predicate  
442 \* returns false and vice versa.  
443 \*  
444 \* @param <T> the type that the predicate queries  
445 \* @param predicate the predicate to not  
446 \* @return the <code>not</code> predicate  
447 \* @throws NullPointerException if the predicate is null  
448 \* @see NotPredicate  
449 \*/  
450 public static <T> Predicate<T> notPredicate(final Predicate<? super T> predicate) {  
451 return NotPredicate.notPredicate(predicate);  
452 }  
453  
454 // Adaptors  
455 //-----------------------------------------------------------------------------  
456  
457 /\*\*  
458 \* Create a new Predicate that wraps a Transformer. The Transformer must  
459 \* return either Boolean.TRUE or Boolean.FALSE otherwise a PredicateException  
460 \* will be thrown.  
461 \*  
462 \* @param <T> the type that the predicate queries  
463 \* @param transformer the transformer to wrap, may not be null  
464 \* @return the transformer wrapping predicate  
465 \* @throws NullPointerException if the transformer is null  
466 \* @see TransformerPredicate  
467 \*/  
468 public static <T> Predicate<T> asPredicate(final Transformer<? super T, Boolean> transformer) {  
469 return TransformerPredicate.transformerPredicate(transformer);  
470 }  
471  
472 // Null handlers  
473 //-----------------------------------------------------------------------------  
474  
475 /\*\*  
476 \* Gets a Predicate that throws an exception if the input object is null,  
477 \* otherwise it calls the specified Predicate. This allows null handling  
478 \* behaviour to be added to Predicates that don't support nulls.  
479 \*  
480 \* @param <T> the type that the predicate queries  
481 \* @param predicate the predicate to wrap, may not be null  
482 \* @return the predicate  
483 \* @throws NullPointerException if the predicate is null.  
484 \* @see NullIsExceptionPredicate  
485 \*/  
486 public static <T> Predicate<T> nullIsExceptionPredicate(final Predicate<? super T> predicate){  
487 return NullIsExceptionPredicate.nullIsExceptionPredicate(predicate);  
488 }  
489  
490 /\*\*  
491 \* Gets a Predicate that returns false if the input object is null, otherwise  
492 \* it calls the specified Predicate. This allows null handling behaviour to  
493 \* be added to Predicates that don't support nulls.  
494 \*  
495 \* @param <T> the type that the predicate queries  
496 \* @param predicate the predicate to wrap, may not be null  
497 \* @return the predicate  
498 \* @throws NullPointerException if the predicate is null.  
499 \* @see NullIsFalsePredicate  
500 \*/  
501 public static <T> Predicate<T> nullIsFalsePredicate(final Predicate<? super T> predicate){  
502 return NullIsFalsePredicate.nullIsFalsePredicate(predicate);  
503 }  
504  
505 /\*\*  
506 \* Gets a Predicate that returns true if the input object is null, otherwise  
507 \* it calls the specified Predicate. This allows null handling behaviour to  
508 \* be added to Predicates that don't support nulls.  
509 \*  
510 \* @param <T> the type that the predicate queries  
511 \* @param predicate the predicate to wrap, may not be null  
512 \* @return the predicate  
513 \* @throws NullPointerException if the predicate is null.  
514 \* @see NullIsTruePredicate  
515 \*/  
516 public static <T> Predicate<T> nullIsTruePredicate(final Predicate<? super T> predicate){  
517 return NullIsTruePredicate.nullIsTruePredicate(predicate);  
518 }  
519  
520 // Transformed  
521 //-----------------------------------------------------------------------  
522 /\*\*  
523 \* Creates a predicate that transforms the input object before passing it  
524 \* to the predicate.  
525 \*  
526 \* @param <T> the type that the predicate queries  
527 \* @param transformer the transformer to call first  
528 \* @param predicate the predicate to call with the result of the transform  
529 \* @return the predicate  
530 \* @throws NullPointerException if the transformer or the predicate is null  
531 \* @see TransformedPredicate  
532 \* @since 3.1  
533 \*/  
534 public static <T> Predicate<T> transformedPredicate(  
535 final Transformer<? super T, ? extends T> transformer, final Predicate<? super T> predicate) {  
536 return TransformedPredicate.transformedPredicate(transformer, predicate);  
537 }  
538  
539}