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.collection;  
018  
019import java.util.ArrayList;  
020import java.util.Collection;  
021import java.util.Collections;  
022import java.util.HashSet;  
023import java.util.LinkedList;  
024import java.util.List;  
025import java.util.Queue;  
026import java.util.Set;  
027  
028import org.apache.commons.collections4.Bag;  
029import org.apache.commons.collections4.MultiSet;  
030import org.apache.commons.collections4.Predicate;  
031import org.apache.commons.collections4.bag.HashBag;  
032import org.apache.commons.collections4.bag.PredicatedBag;  
033import org.apache.commons.collections4.functors.NotNullPredicate;  
034import org.apache.commons.collections4.list.PredicatedList;  
035import org.apache.commons.collections4.multiset.HashMultiSet;  
036import org.apache.commons.collections4.multiset.PredicatedMultiSet;  
037import org.apache.commons.collections4.queue.PredicatedQueue;  
038import org.apache.commons.collections4.set.PredicatedSet;  
039  
040/\*\*  
041 \* Decorates another {@link Collection} to validate that additions  
042 \* match a specified predicate.  
043 \* <p>  
044 \* This collection exists to provide validation for the decorated collection.  
045 \* It is normally created to decorate an empty collection.  
046 \* If an object cannot be added to the collection, an IllegalArgumentException is thrown.  
047 \* </p>  
048 \* <p>  
049 \* One usage would be to ensure that no null entries are added to the collection:  
050 \* </p>  
051 \* <pre>  
052 \* Collection coll = PredicatedCollection.predicatedCollection(new ArrayList(), NotNullPredicate.INSTANCE);  
053 \* </pre>  
054 \* <p>  
055 \* This class is Serializable from Commons Collections 3.1.  
056 \* </p>  
057 \*  
058 \* @param <E> the type of the elements in the collection  
059 \* @since 3.0  
060 \*/  
061public class PredicatedCollection<E> extends AbstractCollectionDecorator<E> {  
062  
063 /\*\* Serialization version \*/  
064 private static final long serialVersionUID = -5259182142076705162L;  
065  
066 /\*\* The predicate to use \*/  
067 protected final Predicate<? super E> predicate;  
068  
069 /\*\*  
070 \* Returns a Builder with the given predicate.  
071 \*  
072 \* @param <E> the element type  
073 \* @param predicate the predicate to use  
074 \* @return a new Builder for predicated collections  
075 \* @since 4.1  
076 \*/  
077 public static <E> Builder<E> builder(final Predicate<? super E> predicate) {  
078 return new Builder<>(predicate);  
079 }  
080  
081 /\*\*  
082 \* Returns a Builder with a NotNullPredicate.  
083 \*  
084 \* @param <E> the element type  
085 \* @return a new Builder for predicated collections that ignores null values.  
086 \* @since 4.1  
087 \*/  
088 public static <E> Builder<E> notNullBuilder() {  
089 return new Builder<>(NotNullPredicate.<E>notNullPredicate());  
090 }  
091  
092 /\*\*  
093 \* Factory method to create a predicated (validating) collection.  
094 \* <p>  
095 \* If there are any elements already in the collection being decorated, they  
096 \* are validated.  
097 \*  
098 \* @param <T> the type of the elements in the collection  
099 \* @param coll the collection to decorate, must not be null  
100 \* @param predicate the predicate to use for validation, must not be null  
101 \* @return a new predicated collection  
102 \* @throws NullPointerException if collection or predicate is null  
103 \* @throws IllegalArgumentException if the collection contains invalid elements  
104 \* @since 4.0  
105 \*/  
106 public static <T> PredicatedCollection<T> predicatedCollection(final Collection<T> coll,  
107 final Predicate<? super T> predicate) {  
108 return new PredicatedCollection<>(coll, predicate);  
109 }  
110  
111 //-----------------------------------------------------------------------  
112 /\*\*  
113 \* Constructor that wraps (not copies).  
114 \* <p>  
115 \* If there are any elements already in the collection being decorated, they  
116 \* are validated.  
117 \*  
118 \* @param coll the collection to decorate, must not be null  
119 \* @param predicate the predicate to use for validation, must not be null  
120 \* @throws NullPointerException if collection or predicate is null  
121 \* @throws IllegalArgumentException if the collection contains invalid elements  
122 \*/  
123 protected PredicatedCollection(final Collection<E> coll, final Predicate<? super E> predicate) {  
124 super(coll);  
125 if (predicate == null) {  
126 throw new NullPointerException("Predicate must not be null.");  
127 }  
128 this.predicate = predicate;  
129 for (final E item : coll) {  
130 validate(item);  
131 }  
132 }  
133  
134 /\*\*  
135 \* Validates the object being added to ensure it matches the predicate.  
136 \* <p>  
137 \* The predicate itself should not throw an exception, but return false to  
138 \* indicate that the object cannot be added.  
139 \*  
140 \* @param object the object being added  
141 \* @throws IllegalArgumentException if the add is invalid  
142 \*/  
143 protected void validate(final E object) {  
144 if (predicate.evaluate(object) == false) {  
145 throw new IllegalArgumentException("Cannot add Object '" + object + "' - Predicate '" +  
146 predicate + "' rejected it");  
147 }  
148 }  
149  
150 //-----------------------------------------------------------------------  
151 /\*\*  
152 \* Override to validate the object being added to ensure it matches  
153 \* the predicate.  
154 \*  
155 \* @param object the object being added  
156 \* @return the result of adding to the underlying collection  
157 \* @throws IllegalArgumentException if the add is invalid  
158 \*/  
159 @Override  
160 public boolean add(final E object) {  
161 validate(object);  
162 return decorated().add(object);  
163 }  
164  
165 /\*\*  
166 \* Override to validate the objects being added to ensure they match  
167 \* the predicate. If any one fails, no update is made to the underlying  
168 \* collection.  
169 \*  
170 \* @param coll the collection being added  
171 \* @return the result of adding to the underlying collection  
172 \* @throws IllegalArgumentException if the add is invalid  
173 \*/  
174 @Override  
175 public boolean addAll(final Collection<? extends E> coll) {  
176 for (final E item : coll) {  
177 validate(item);  
178 }  
179 return decorated().addAll(coll);  
180 }  
181  
182 /\*\*  
183 \* Builder for creating predicated collections.  
184 \* <p>  
185 \* Create a Builder with a predicate to validate elements against, then add any elements  
186 \* to the builder. Elements that fail the predicate will be added to a rejected list.  
187 \* Finally create or decorate a collection using the createPredicated[List,Set,Bag,Queue] methods.  
188 \* <p>  
189 \* An example:  
190 \* <pre>  
191 \* Predicate<String> predicate = NotNullPredicate.notNullPredicate();  
192 \* PredicatedCollectionBuilder<String> builder = PredicatedCollection.builder(predicate);  
193 \* builder.add("item1");  
194 \* builder.add(null);  
195 \* builder.add("item2");  
196 \* List<String> predicatedList = builder.createPredicatedList();  
197 \* </pre>  
198 \* <p>  
199 \* At the end of the code fragment above predicatedList is protected by the predicate supplied  
200 \* to the builder and it contains item1 and item2.  
201 \* <p>  
202 \* More elements can be added to the builder once a predicated collection has been created,  
203 \* but these elements will not be reflected in already created collections.  
204 \*  
205 \* @param <E> the element type  
206 \* @since 4.1  
207 \*/  
208 public static class Builder<E> {  
209  
210 /\*\* The predicate to use. \*/  
211 private final Predicate<? super E> predicate;  
212  
213 /\*\* The buffer containing valid elements. \*/  
214 private final List<E> accepted = new ArrayList<>();  
215  
216 /\*\* The buffer containing rejected elements. \*/  
217 private final List<E> rejected = new ArrayList<>();  
218  
219 // -----------------------------------------------------------------------  
220 /\*\*  
221 \* Constructs a PredicatedCollectionBuilder with the specified Predicate.  
222 \*  
223 \* @param predicate the predicate to use  
224 \* @throws NullPointerException if predicate is null  
225 \*/  
226 public Builder(final Predicate<? super E> predicate) {  
227 if (predicate == null) {  
228 throw new NullPointerException("Predicate must not be null");  
229 }  
230 this.predicate = predicate;  
231 }  
232  
233 /\*\*  
234 \* Adds the item to the builder.  
235 \* <p>  
236 \* If the predicate is true, it is added to the list of accepted elements,  
237 \* otherwise it is added to the rejected list.  
238 \*  
239 \* @param item the element to add  
240 \* @return the PredicatedCollectionBuilder.  
241 \*/  
242 public Builder<E> add(final E item) {  
243 if (predicate.evaluate(item)) {  
244 accepted.add(item);  
245 } else {  
246 rejected.add(item);  
247 }  
248 return this;  
249 }  
250  
251 /\*\*  
252 \* Adds all elements from the given collection to the builder.  
253 \* <p>  
254 \* All elements for which the predicate evaluates to true will be added to the  
255 \* list of accepted elements, otherwise they are added to the rejected list.  
256 \*  
257 \* @param items the elements to add to the builder  
258 \* @return the PredicatedCollectionBuilder.  
259 \*/  
260 public Builder<E> addAll(final Collection<? extends E> items) {  
261 if (items != null) {  
262 for (final E item : items) {  
263 add(item);  
264 }  
265 }  
266 return this;  
267 }  
268  
269 /\*\*  
270 \* Create a new predicated list filled with the accepted elements.  
271 \* <p>  
272 \* The builder is not modified by this method, so it is possible to create more collections  
273 \* or add more elements afterwards. Further changes will not propagate to the returned list.  
274 \*  
275 \* @return a new predicated list.  
276 \*/  
277 public List<E> createPredicatedList() {  
278 return createPredicatedList(new ArrayList<E>());  
279 }  
280  
281 /\*\*  
282 \* Decorates the given list with validating behavior using the predicate. All accepted elements  
283 \* are appended to the list. If the list already contains elements, they are validated.  
284 \* <p>  
285 \* The builder is not modified by this method, so it is possible to create more collections  
286 \* or add more elements afterwards. Further changes will not propagate to the returned list.  
287 \*  
288 \* @param list the List to decorate, must not be null  
289 \* @return the decorated list.  
290 \* @throws NullPointerException if list is null  
291 \* @throws IllegalArgumentException if list contains invalid elements  
292 \*/  
293 public List<E> createPredicatedList(final List<E> list) {  
294 if (list == null) {  
295 throw new NullPointerException("List must not be null.");  
296 }  
297 final List<E> predicatedList = PredicatedList.predicatedList(list, predicate);  
298 predicatedList.addAll(accepted);  
299 return predicatedList;  
300 }  
301  
302 /\*\*  
303 \* Create a new predicated set filled with the accepted elements.  
304 \* <p>  
305 \* The builder is not modified by this method, so it is possible to create more collections  
306 \* or add more elements afterwards. Further changes will not propagate to the returned set.  
307 \*  
308 \* @return a new predicated set.  
309 \*/  
310 public Set<E> createPredicatedSet() {  
311 return createPredicatedSet(new HashSet<E>());  
312 }  
313  
314 /\*\*  
315 \* Decorates the given list with validating behavior using the predicate. All accepted elements  
316 \* are appended to the set. If the set already contains elements, they are validated.  
317 \* <p>  
318 \* The builder is not modified by this method, so it is possible to create more collections  
319 \* or add more elements afterwards. Further changes will not propagate to the returned set.  
320 \*  
321 \* @param set the set to decorate, must not be null  
322 \* @return the decorated set.  
323 \* @throws NullPointerException if set is null  
324 \* @throws IllegalArgumentException if set contains invalid elements  
325 \*/  
326 public Set<E> createPredicatedSet(final Set<E> set) {  
327 if (set == null) {  
328 throw new NullPointerException("Set must not be null.");  
329 }  
330 final PredicatedSet<E> predicatedSet = PredicatedSet.predicatedSet(set, predicate);  
331 predicatedSet.addAll(accepted);  
332 return predicatedSet;  
333 }  
334  
335 /\*\*  
336 \* Create a new predicated multiset filled with the accepted elements.  
337 \* <p>  
338 \* The builder is not modified by this method, so it is possible to create more collections  
339 \* or add more elements afterwards. Further changes will not propagate to the returned multiset.  
340 \*  
341 \* @return a new predicated multiset.  
342 \*/  
343 public MultiSet<E> createPredicatedMultiSet() {  
344 return createPredicatedMultiSet(new HashMultiSet<E>());  
345 }  
346  
347 /\*\*  
348 \* Decorates the given multiset with validating behavior using the predicate. All accepted elements  
349 \* are appended to the multiset. If the multiset already contains elements, they are validated.  
350 \* <p>  
351 \* The builder is not modified by this method, so it is possible to create more collections  
352 \* or add more elements afterwards. Further changes will not propagate to the returned multiset.  
353 \*  
354 \* @param multiset the multiset to decorate, must not be null  
355 \* @return the decorated multiset.  
356 \* @throws NullPointerException if multiset is null  
357 \* @throws IllegalArgumentException if multiset contains invalid elements  
358 \*/  
359 public MultiSet<E> createPredicatedMultiSet(final MultiSet<E> multiset) {  
360 if (multiset == null) {  
361 throw new NullPointerException("MultiSet must not be null.");  
362 }  
363 final PredicatedMultiSet<E> predicatedMultiSet =  
364 PredicatedMultiSet.predicatedMultiSet(multiset, predicate);  
365 predicatedMultiSet.addAll(accepted);  
366 return predicatedMultiSet;  
367 }  
368  
369 /\*\*  
370 \* Create a new predicated bag filled with the accepted elements.  
371 \* <p>  
372 \* The builder is not modified by this method, so it is possible to create more collections  
373 \* or add more elements afterwards. Further changes will not propagate to the returned bag.  
374 \*  
375 \* @return a new predicated bag.  
376 \*/  
377 public Bag<E> createPredicatedBag() {  
378 return createPredicatedBag(new HashBag<E>());  
379 }  
380  
381 /\*\*  
382 \* Decorates the given bag with validating behavior using the predicate. All accepted elements  
383 \* are appended to the bag. If the bag already contains elements, they are validated.  
384 \* <p>  
385 \* The builder is not modified by this method, so it is possible to create more collections  
386 \* or add more elements afterwards. Further changes will not propagate to the returned bag.  
387 \*  
388 \* @param bag the bag to decorate, must not be null  
389 \* @return the decorated bag.  
390 \* @throws NullPointerException if bag is null  
391 \* @throws IllegalArgumentException if bag contains invalid elements  
392 \*/  
393 public Bag<E> createPredicatedBag(final Bag<E> bag) {  
394 if (bag == null) {  
395 throw new NullPointerException("Bag must not be null.");  
396 }  
397 final PredicatedBag<E> predicatedBag = PredicatedBag.predicatedBag(bag, predicate);  
398 predicatedBag.addAll(accepted);  
399 return predicatedBag;  
400 }  
401  
402 /\*\*  
403 \* Create a new predicated queue filled with the accepted elements.  
404 \* <p>  
405 \* The builder is not modified by this method, so it is possible to create more collections  
406 \* or add more elements afterwards. Further changes will not propagate to the returned queue.  
407 \*  
408 \* @return a new predicated queue.  
409 \*/  
410 public Queue<E> createPredicatedQueue() {  
411 return createPredicatedQueue(new LinkedList<E>());  
412 }  
413  
414 /\*\*  
415 \* Decorates the given queue with validating behavior using the predicate. All accepted elements  
416 \* are appended to the queue. If the queue already contains elements, they are validated.  
417 \* <p>  
418 \* The builder is not modified by this method, so it is possible to create more collections  
419 \* or add more elements afterwards. Further changes will not propagate to the returned queue.  
420 \*  
421 \* @param queue the queue to decorate, must not be null  
422 \* @return the decorated queue.  
423 \* @throws NullPointerException if queue is null  
424 \* @throws IllegalArgumentException if queue contains invalid elements  
425 \*/  
426 public Queue<E> createPredicatedQueue(final Queue<E> queue) {  
427 if (queue == null) {  
428 throw new NullPointerException("queue must not be null");  
429 }  
430 final PredicatedQueue<E> predicatedQueue = PredicatedQueue.predicatedQueue(queue, predicate);  
431 predicatedQueue.addAll(accepted);  
432 return predicatedQueue;  
433 }  
434  
435 /\*\*  
436 \* Returns an unmodifiable collection containing all rejected elements.  
437 \*  
438 \* @return an unmodifiable collection  
439 \*/  
440 public Collection<E> rejectedElements() {  
441 return Collections.unmodifiableCollection(rejected);  
442 }  
443  
444 }  
445  
446}