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.List;  
022  
023import org.apache.commons.collections4.Transformer;  
024  
025/\*\*  
026 \* Decorates another {@link Collection} to transform objects that are added.  
027 \* <p>  
028 \* The add methods are affected by this class.  
029 \* Thus objects must be removed or searched for using their transformed form.  
030 \* For example, if the transformation converts Strings to Integers, you must  
031 \* use the Integer form to remove objects.  
032 \* </p>  
033 \* <p>  
034 \* This class is Serializable from Commons Collections 3.1.  
035 \* </p>  
036 \*  
037 \* @param <E> the type of the elements in the collection  
038 \* @since 3.0  
039 \*/  
040public class TransformedCollection<E> extends AbstractCollectionDecorator<E> {  
041  
042 /\*\* Serialization version \*/  
043 private static final long serialVersionUID = 8692300188161871514L;  
044  
045 /\*\* The transformer to use \*/  
046 protected final Transformer<? super E, ? extends E> transformer;  
047  
048 /\*\*  
049 \* Factory method to create a transforming collection.  
050 \* <p>  
051 \* If there are any elements already in the collection being decorated, they  
052 \* are NOT transformed.  
053 \* Contrast this with {@link #transformedCollection(Collection, Transformer)}.  
054 \*  
055 \* @param <E> the type of the elements in the collection  
056 \* @param coll the collection to decorate, must not be null  
057 \* @param transformer the transformer to use for conversion, must not be null  
058 \* @return a new transformed collection  
059 \* @throws NullPointerException if collection or transformer is null  
060 \* @since 4.0  
061 \*/  
062 public static <E> TransformedCollection<E> transformingCollection(final Collection<E> coll,  
063 final Transformer<? super E, ? extends E> transformer) {  
064 return new TransformedCollection<>(coll, transformer);  
065 }  
066  
067 /\*\*  
068 \* Factory method to create a transforming collection that will transform  
069 \* existing contents of the specified collection.  
070 \* <p>  
071 \* If there are any elements already in the collection being decorated, they  
072 \* will be transformed by this method.  
073 \* Contrast this with {@link #transformingCollection(Collection, Transformer)}.  
074 \*  
075 \* @param <E> the type of the elements in the collection  
076 \* @param collection the collection to decorate, must not be null  
077 \* @param transformer the transformer to use for conversion, must not be null  
078 \* @return a new transformed Collection  
079 \* @throws NullPointerException if collection or transformer is null  
080 \* @since 4.0  
081 \*/  
082 public static <E> TransformedCollection<E> transformedCollection(final Collection<E> collection,  
083 final Transformer<? super E, ? extends E> transformer) {  
084  
085 final TransformedCollection<E> decorated = new TransformedCollection<>(collection, transformer);  
086 // null collection & transformer are disallowed by the constructor call above  
087 if (collection.size() > 0) {  
088 @SuppressWarnings("unchecked") // collection is of type E  
089 final E[] values = (E[]) collection.toArray(); // NOPMD - false positive for generics  
090 collection.clear();  
091 for (final E value : values) {  
092 decorated.decorated().add(transformer.transform(value));  
093 }  
094 }  
095 return decorated;  
096 }  
097  
098 //-----------------------------------------------------------------------  
099 /\*\*  
100 \* Constructor that wraps (not copies).  
101 \* <p>  
102 \* If there are any elements already in the collection being decorated, they  
103 \* are NOT transformed.  
104 \*  
105 \* @param coll the collection to decorate, must not be null  
106 \* @param transformer the transformer to use for conversion, must not be null  
107 \* @throws NullPointerException if collection or transformer is null  
108 \*/  
109 protected TransformedCollection(final Collection<E> coll, final Transformer<? super E, ? extends E> transformer) {  
110 super(coll);  
111 if (transformer == null) {  
112 throw new NullPointerException("Transformer must not be null");  
113 }  
114 this.transformer = transformer;  
115 }  
116  
117 /\*\*  
118 \* Transforms an object.  
119 \* <p>  
120 \* The transformer itself may throw an exception if necessary.  
121 \*  
122 \* @param object the object to transform  
123 \* @return a transformed object  
124 \*/  
125 protected E transform(final E object) {  
126 return transformer.transform(object);  
127 }  
128  
129 /\*\*  
130 \* Transforms a collection.  
131 \* <p>  
132 \* The transformer itself may throw an exception if necessary.  
133 \*  
134 \* @param coll the collection to transform  
135 \* @return a transformed object  
136 \*/  
137 protected Collection<E> transform(final Collection<? extends E> coll) {  
138 final List<E> list = new ArrayList<>(coll.size());  
139 for (final E item : coll) {  
140 list.add(transform(item));  
141 }  
142 return list;  
143 }  
144  
145 //-----------------------------------------------------------------------  
146 @Override  
147 public boolean add(final E object) {  
148 return decorated().add(transform(object));  
149 }  
150  
151 @Override  
152 public boolean addAll(final Collection<? extends E> coll) {  
153 return decorated().addAll(transform(coll));  
154 }  
155  
156}