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.iterators;  
018  
019import java.util.Iterator;  
020  
021import org.apache.commons.collections4.Transformer;  
022  
023/\*\*  
024 \* Decorates an iterator such that each element returned is transformed.  
025 \*  
026 \* @since 1.0  
027 \*/  
028public class TransformIterator<I, O> implements Iterator<O> {  
029  
030 /\*\* The iterator being used \*/  
031 private Iterator<? extends I> iterator;  
032 /\*\* The transformer being used \*/  
033 private Transformer<? super I, ? extends O> transformer;  
034  
035 //-----------------------------------------------------------------------  
036 /\*\*  
037 \* Constructs a new <code>TransformIterator</code> that will not function  
038 \* until the {@link #setIterator(Iterator) setIterator} and  
039 \* {@link #setTransformer(Transformer)} methods are invoked.  
040 \*/  
041 public TransformIterator() {  
042 super();  
043 }  
044  
045 /\*\*  
046 \* Constructs a new <code>TransformIterator</code> that won't transform  
047 \* elements from the given iterator.  
048 \*  
049 \* @param iterator the iterator to use  
050 \*/  
051 public TransformIterator(final Iterator<? extends I> iterator) {  
052 super();  
053 this.iterator = iterator;  
054 }  
055  
056 /\*\*  
057 \* Constructs a new <code>TransformIterator</code> that will use the  
058 \* given iterator and transformer. If the given transformer is null,  
059 \* then objects will not be transformed.  
060 \*  
061 \* @param iterator the iterator to use  
062 \* @param transformer the transformer to use  
063 \*/  
064 public TransformIterator(final Iterator<? extends I> iterator,  
065 final Transformer<? super I, ? extends O> transformer) {  
066 super();  
067 this.iterator = iterator;  
068 this.transformer = transformer;  
069 }  
070  
071 //-----------------------------------------------------------------------  
072 @Override  
073 public boolean hasNext() {  
074 return iterator.hasNext();  
075 }  
076  
077 /\*\*  
078 \* Gets the next object from the iteration, transforming it using the  
079 \* current transformer. If the transformer is null, no transformation  
080 \* occurs and the object from the iterator is returned directly.  
081 \*  
082 \* @return the next object  
083 \* @throws java.util.NoSuchElementException if there are no more elements  
084 \*/  
085 @Override  
086 public O next() {  
087 return transform(iterator.next());  
088 }  
089  
090 @Override  
091 public void remove() {  
092 iterator.remove();  
093 }  
094  
095 //-----------------------------------------------------------------------  
096 /\*\*  
097 \* Gets the iterator this iterator is using.  
098 \*  
099 \* @return the iterator.  
100 \*/  
101 public Iterator<? extends I> getIterator() {  
102 return iterator;  
103 }  
104  
105 /\*\*  
106 \* Sets the iterator for this iterator to use.  
107 \* If iteration has started, this effectively resets the iterator.  
108 \*  
109 \* @param iterator the iterator to use  
110 \*/  
111 public void setIterator(final Iterator<? extends I> iterator) {  
112 this.iterator = iterator;  
113 }  
114  
115 //-----------------------------------------------------------------------  
116 /\*\*  
117 \* Gets the transformer this iterator is using.  
118 \*  
119 \* @return the transformer.  
120 \*/  
121 public Transformer<? super I, ? extends O> getTransformer() {  
122 return transformer;  
123 }  
124  
125 /\*\*  
126 \* Sets the transformer this the iterator to use.  
127 \* A null transformer is a no-op transformer.  
128 \*  
129 \* @param transformer the transformer to use  
130 \*/  
131 public void setTransformer(final Transformer<? super I, ? extends O> transformer) {  
132 this.transformer = transformer;  
133 }  
134  
135 //-----------------------------------------------------------------------  
136 /\*\*  
137 \* Transforms the given object using the transformer.  
138 \* If the transformer is null, the original object is returned as-is.  
139 \*  
140 \* @param source the object to transform  
141 \* @return the transformed object  
142 \*/  
143 protected O transform(final I source) {  
144 return transformer.transform(source);  
145 }  
146}