001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with this  
004 \* work for additional information regarding copyright ownership. The ASF  
005 \* licenses this file to You under the Apache License, Version 2.0 (the  
006 \* "License"); you may not use this file except in compliance with the License.  
007 \* You may obtain a copy of the License at  
008 \* http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law  
009 \* or agreed to in writing, software distributed under the License is  
010 \* distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY  
011 \* KIND, either express or implied. See the License for the specific language  
012 \* governing permissions and limitations under the License.  
013 \*/  
014package org.apache.commons.collections4.iterators;  
015  
016import java.util.Iterator;  
017  
018/\*\*  
019 \* Decorates another iterator to skip the first N elements.  
020 \* <p>  
021 \* In case an offset parameter other than 0 is provided, the decorated  
022 \* iterator is immediately advanced to this position, skipping all elements  
023 \* before that position.  
024 \* </p>  
025 \*  
026 \* @since 4.1  
027 \*/  
028public class SkippingIterator<E> extends AbstractIteratorDecorator<E> {  
029  
030 /\*\* The offset to bound the first element return \*/  
031 private final long offset;  
032  
033 /\*\* The position of the current element \*/  
034 private long pos;  
035  
036 //-----------------------------------------------------------------------  
037  
038 /\*\*  
039 \* Decorates the specified iterator to skip all elements until the iterator  
040 \* reaches the position at {@code offset}.  
041 \* <p>  
042 \* The iterator is immediately advanced until it reaches the position at {@code offset},  
043 \* incurring O(n) time.  
044 \*  
045 \* @param iterator the iterator to be decorated  
046 \* @param offset the index of the first element of the decorated iterator to return  
047 \* @throws NullPointerException if iterator is null  
048 \* @throws IllegalArgumentException if offset is negative  
049 \*/  
050 public SkippingIterator(final Iterator<E> iterator, final long offset) {  
051 super(iterator);  
052  
053 if (offset < 0) {  
054 throw new IllegalArgumentException("Offset parameter must not be negative.");  
055 }  
056  
057 this.offset = offset;  
058 this.pos = 0;  
059 init();  
060 }  
061  
062 /\*\*  
063 \* Skips the given number of elements.  
064 \*/  
065 private void init() {  
066 while (pos < offset && hasNext()) {  
067 next();  
068 }  
069 }  
070  
071 //-----------------------------------------------------------------------  
072  
073 @Override  
074 public E next() {  
075 final E next = super.next();  
076 pos++;  
077 return next;  
078 }  
079  
080 /\*\*  
081 \* {@inheritDoc}  
082 \* <p>  
083 \* In case an offset other than 0 was specified, the underlying iterator will be advanced  
084 \* to this position upon creation. A call to {@link #remove()} will still result in an  
085 \* {@link IllegalStateException} if no explicit call to {@link #next()} has been made prior  
086 \* to calling {@link #remove()}.  
087 \*/  
088 @Override  
089 public void remove() {  
090 if (pos <= offset) {  
091 throw new IllegalStateException("remove() can not be called before calling next()");  
092 }  
093 super.remove();  
094 }  
095  
096}