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  
021/\*\*  
022 \* Provides basic behaviour for decorating an iterator with extra functionality  
023 \* without committing the generic type of the Iterator implementation.  
024 \* <p>  
025 \* All methods are forwarded to the decorated iterator.  
026 \*  
027 \* @since 4.0  
028 \*/  
029public abstract class AbstractUntypedIteratorDecorator<I, O> implements Iterator<O> {  
030  
031 /\*\* The iterator being decorated \*/  
032 private final Iterator<I> iterator;  
033  
034 /\*\*  
035 \* Create a new AbstractUntypedIteratorDecorator.  
036 \*  
037 \* @param iterator the iterator to decorate  
038 \* @throws NullPointerException if the iterator is null  
039 \*/  
040 protected AbstractUntypedIteratorDecorator(final Iterator<I> iterator) {  
041 super();  
042 if (iterator == null) {  
043 throw new NullPointerException("Iterator must not be null");  
044 }  
045 this.iterator = iterator;  
046 }  
047  
048 /\*\*  
049 \* Gets the iterator being decorated.  
050 \*  
051 \* @return the decorated iterator  
052 \*/  
053 protected Iterator<I> getIterator() {  
054 return iterator;  
055 }  
056  
057 @Override  
058 public boolean hasNext() {  
059 return iterator.hasNext();  
060 }  
061  
062 @Override  
063 public void remove() {  
064 iterator.remove();  
065 }  
066  
067}