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;  
020import java.util.NoSuchElementException;  
021  
022import org.apache.commons.collections4.Predicate;  
023  
024/\*\*  
025 \* Decorates another {@link Iterator} using a predicate to filter elements.  
026 \* <p>  
027 \* This iterator decorates the underlying iterator, only allowing through  
028 \* those elements that match the specified {@link Predicate Predicate}.  
029 \*  
030 \* @since 1.0  
031 \*/  
032public class FilterIterator<E> implements Iterator<E> {  
033  
034 /\*\* The iterator being used \*/  
035 private Iterator<? extends E> iterator;  
036 /\*\* The predicate being used \*/  
037 private Predicate<? super E> predicate;  
038 /\*\* The next object in the iteration \*/  
039 private E nextObject;  
040 /\*\* Whether the next object has been calculated yet \*/  
041 private boolean nextObjectSet = false;  
042  
043 //-----------------------------------------------------------------------  
044 /\*\*  
045 \* Constructs a new <code>FilterIterator</code> that will not function  
046 \* until {@link #setIterator(Iterator) setIterator} is invoked.  
047 \*/  
048 public FilterIterator() {  
049 super();  
050 }  
051  
052 /\*\*  
053 \* Constructs a new <code>FilterIterator</code> that will not function  
054 \* until {@link #setPredicate(Predicate) setPredicate} is invoked.  
055 \*  
056 \* @param iterator the iterator to use  
057 \*/  
058 public FilterIterator(final Iterator<? extends E> iterator) {  
059 super();  
060 this.iterator = iterator;  
061 }  
062  
063 /\*\*  
064 \* Constructs a new <code>FilterIterator</code> that will use the  
065 \* given iterator and predicate.  
066 \*  
067 \* @param iterator the iterator to use  
068 \* @param predicate the predicate to use  
069 \*/  
070 public FilterIterator(final Iterator<? extends E> iterator, final Predicate<? super E> predicate) {  
071 super();  
072 this.iterator = iterator;  
073 this.predicate = predicate;  
074 }  
075  
076 //-----------------------------------------------------------------------  
077 /\*\*  
078 \* Returns true if the underlying iterator contains an object that  
079 \* matches the predicate.  
080 \*  
081 \* @return true if there is another object that matches the predicate  
082 \* @throws NullPointerException if either the iterator or predicate are null  
083 \*/  
084 @Override  
085 public boolean hasNext() {  
086 return nextObjectSet || setNextObject();  
087 }  
088  
089 /\*\*  
090 \* Returns the next object that matches the predicate.  
091 \*  
092 \* @return the next object which matches the given predicate  
093 \* @throws NullPointerException if either the iterator or predicate are null  
094 \* @throws NoSuchElementException if there are no more elements that  
095 \* match the predicate  
096 \*/  
097 @Override  
098 public E next() {  
099 if (!nextObjectSet && !setNextObject()) {  
100 throw new NoSuchElementException();  
101 }  
102 nextObjectSet = false;  
103 return nextObject;  
104 }  
105  
106 /\*\*  
107 \* Removes from the underlying collection of the base iterator the last  
108 \* element returned by this iterator.  
109 \* This method can only be called  
110 \* if <code>next()</code> was called, but not after  
111 \* <code>hasNext()</code>, because the <code>hasNext()</code> call  
112 \* changes the base iterator.  
113 \*  
114 \* @throws IllegalStateException if <code>hasNext()</code> has already  
115 \* been called.  
116 \*/  
117 @Override  
118 public void remove() {  
119 if (nextObjectSet) {  
120 throw new IllegalStateException("remove() cannot be called");  
121 }  
122 iterator.remove();  
123 }  
124  
125 //-----------------------------------------------------------------------  
126 /\*\*  
127 \* Gets the iterator this iterator is using.  
128 \*  
129 \* @return the iterator  
130 \*/  
131 public Iterator<? extends E> getIterator() {  
132 return iterator;  
133 }  
134  
135 /\*\*  
136 \* Sets the iterator for this iterator to use.  
137 \* If iteration has started, this effectively resets the iterator.  
138 \*  
139 \* @param iterator the iterator to use  
140 \*/  
141 public void setIterator(final Iterator<? extends E> iterator) {  
142 this.iterator = iterator;  
143 nextObject = null;  
144 nextObjectSet = false;  
145 }  
146  
147 //-----------------------------------------------------------------------  
148 /\*\*  
149 \* Gets the predicate this iterator is using.  
150 \*  
151 \* @return the predicate  
152 \*/  
153 public Predicate<? super E> getPredicate() {  
154 return predicate;  
155 }  
156  
157 /\*\*  
158 \* Sets the predicate this the iterator to use.  
159 \*  
160 \* @param predicate the predicate to use  
161 \*/  
162 public void setPredicate(final Predicate<? super E> predicate) {  
163 this.predicate = predicate;  
164 nextObject = null;  
165 nextObjectSet = false;  
166 }  
167  
168 //-----------------------------------------------------------------------  
169 /\*\*  
170 \* Set nextObject to the next object. If there are no more  
171 \* objects then return false. Otherwise, return true.  
172 \*/  
173 private boolean setNextObject() {  
174 while (iterator.hasNext()) {  
175 final E object = iterator.next();  
176 if (predicate.evaluate(object)) {  
177 nextObject = object;  
178 nextObjectSet = true;  
179 return true;  
180 }  
181 }  
182 return false;  
183 }  
184  
185}