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.list;  
018  
019import java.util.Collection;  
020import java.util.List;  
021import java.util.ListIterator;  
022  
023import org.apache.commons.collections4.Predicate;  
024import org.apache.commons.collections4.collection.PredicatedCollection;  
025import org.apache.commons.collections4.iterators.AbstractListIteratorDecorator;  
026  
027/\*\*  
028 \* Decorates another <code>List</code> to validate that all additions  
029 \* match a specified predicate.  
030 \* <p>  
031 \* This list exists to provide validation for the decorated list.  
032 \* It is normally created to decorate an empty list.  
033 \* If an object cannot be added to the list, an IllegalArgumentException is thrown.  
034 \* </p>  
035 \* <p>  
036 \* One usage would be to ensure that no null entries are added to the list.  
037 \* </p>  
038 \* <pre>  
039 \* {@code  
040 \* List<String> list =  
041 \* PredicatedList.predicatedList(new ArrayList<String>(), PredicateUtils.notNullPredicate());  
042 \* }  
043 \* </pre>  
044 \* <p>  
045 \* This class is Serializable from Commons Collections 3.1.  
046 \* </p>  
047 \*  
048 \* @since 3.0  
049 \*/  
050public class PredicatedList<E> extends PredicatedCollection<E> implements List<E> {  
051  
052 /\*\* Serialization version \*/  
053 private static final long serialVersionUID = -5722039223898659102L;  
054  
055 /\*\*  
056 \* Factory method to create a predicated (validating) list.  
057 \* <p>  
058 \* If there are any elements already in the list being decorated, they  
059 \* are validated.  
060 \*  
061 \* @param <T> the type of the elements in the list  
062 \* @param list the list to decorate, must not be null  
063 \* @param predicate the predicate to use for validation, must not be null  
064 \* @return a new predicated list  
065 \* @throws NullPointerException if list or predicate is null  
066 \* @throws IllegalArgumentException if the list contains invalid elements  
067 \* @since 4.0  
068 \*/  
069 public static <T> PredicatedList<T> predicatedList(final List<T> list, final Predicate<? super T> predicate) {  
070 return new PredicatedList<>(list, predicate);  
071 }  
072  
073 //-----------------------------------------------------------------------  
074 /\*\*  
075 \* Constructor that wraps (not copies).  
076 \* <p>  
077 \* If there are any elements already in the list being decorated, they  
078 \* are validated.  
079 \*  
080 \* @param list the list to decorate, must not be null  
081 \* @param predicate the predicate to use for validation, must not be null  
082 \* @throws NullPointerException if list or predicate is null  
083 \* @throws IllegalArgumentException if the list contains invalid elements  
084 \*/  
085 protected PredicatedList(final List<E> list, final Predicate<? super E> predicate) {  
086 super(list, predicate);  
087 }  
088  
089 /\*\*  
090 \* Gets the list being decorated.  
091 \*  
092 \* @return the decorated list  
093 \*/  
094 @Override  
095 protected List<E> decorated() {  
096 return (List<E>) super.decorated();  
097 }  
098  
099 @Override  
100 public boolean equals(final Object object) {  
101 return object == this || decorated().equals(object);  
102 }  
103  
104 @Override  
105 public int hashCode() {  
106 return decorated().hashCode();  
107 }  
108  
109 //-----------------------------------------------------------------------  
110  
111 @Override  
112 public E get(final int index) {  
113 return decorated().get(index);  
114 }  
115  
116 @Override  
117 public int indexOf(final Object object) {  
118 return decorated().indexOf(object);  
119 }  
120  
121 @Override  
122 public int lastIndexOf(final Object object) {  
123 return decorated().lastIndexOf(object);  
124 }  
125  
126 @Override  
127 public E remove(final int index) {  
128 return decorated().remove(index);  
129 }  
130  
131 //-----------------------------------------------------------------------  
132  
133 @Override  
134 public void add(final int index, final E object) {  
135 validate(object);  
136 decorated().add(index, object);  
137 }  
138  
139 @Override  
140 public boolean addAll(final int index, final Collection<? extends E> coll) {  
141 for (final E aColl : coll) {  
142 validate(aColl);  
143 }  
144 return decorated().addAll(index, coll);  
145 }  
146  
147 @Override  
148 public ListIterator<E> listIterator() {  
149 return listIterator(0);  
150 }  
151  
152 @Override  
153 public ListIterator<E> listIterator(final int i) {  
154 return new PredicatedListIterator(decorated().listIterator(i));  
155 }  
156  
157 @Override  
158 public E set(final int index, final E object) {  
159 validate(object);  
160 return decorated().set(index, object);  
161 }  
162  
163 @Override  
164 public List<E> subList(final int fromIndex, final int toIndex) {  
165 final List<E> sub = decorated().subList(fromIndex, toIndex);  
166 return new PredicatedList<>(sub, predicate);  
167 }  
168  
169 /\*\*  
170 \* Inner class Iterator for the PredicatedList  
171 \*/  
172 protected class PredicatedListIterator extends AbstractListIteratorDecorator<E> {  
173  
174 /\*\*  
175 \* Create a new predicated list iterator.  
176 \*  
177 \* @param iterator the list iterator to decorate  
178 \*/  
179 protected PredicatedListIterator(final ListIterator<E> iterator) {  
180 super(iterator);  
181 }  
182  
183 @Override  
184 public void add(final E object) {  
185 validate(object);  
186 getListIterator().add(object);  
187 }  
188  
189 @Override  
190 public void set(final E object) {  
191 validate(object);  
192 getListIterator().set(object);  
193 }  
194 }  
195  
196}