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.Iterator;  
021import java.util.List;  
022import java.util.ListIterator;  
023import java.util.function.Predicate;  
024  
025import org.apache.commons.collections4.BoundedCollection;  
026import org.apache.commons.collections4.iterators.AbstractListIteratorDecorator;  
027import org.apache.commons.collections4.iterators.UnmodifiableIterator;  
028  
029/\*\*  
030 \* Decorates another <code>List</code> to fix the size preventing add/remove.  
031 \* <p>  
032 \* The add, remove, clear and retain operations are unsupported.  
033 \* The set method is allowed (as it doesn't change the list size).  
034 \* </p>  
035 \* <p>  
036 \* NOTE:  
037 \* Modifying the decorated list directly would results in influencing the outcome  
038 \* of method calls on this object. For example, the bounds of this list would reflect  
039 \* a newly added object to the underlying list.  
040 \* </p>  
041 \* <p>  
042 \* This class is Serializable from Commons Collections 3.1.  
043 \* </p>  
044 \*  
045 \* @param <E> the type of elements in this collection  
046 \* @since 3.0  
047 \*/  
048public class FixedSizeList<E>  
049 extends AbstractSerializableListDecorator<E>  
050 implements BoundedCollection<E> {  
051  
052 /\*\* Serialization version \*/  
053 private static final long serialVersionUID = -2218010673611160319L;  
054  
055 /\*\*  
056 \* Factory method to create a fixed size list.  
057 \*  
058 \* @param <E> the type of the elements in the list  
059 \* @param list the list to decorate, must not be null  
060 \* @return a new fixed size list  
061 \* @throws NullPointerException if list is null  
062 \* @since 4.0  
063 \*/  
064 public static <E> FixedSizeList<E> fixedSizeList(final List<E> list) {  
065 return new FixedSizeList<>(list);  
066 }  
067  
068 //-----------------------------------------------------------------------  
069 /\*\*  
070 \* Constructor that wraps (not copies).  
071 \*  
072 \* @param list the list to decorate, must not be null  
073 \* @throws NullPointerException if list is null  
074 \*/  
075 protected FixedSizeList(final List<E> list) {  
076 super(list);  
077 }  
078  
079 //-----------------------------------------------------------------------  
080 @Override  
081 public boolean add(final E object) {  
082 throw unsupportedOperationException();  
083 }  
084  
085 @Override  
086 public void add(final int index, final E object) {  
087 throw unsupportedOperationException();  
088 }  
089  
090 @Override  
091 public boolean addAll(final Collection<? extends E> coll) {  
092 throw unsupportedOperationException();  
093 }  
094  
095 @Override  
096 public boolean addAll(final int index, final Collection<? extends E> coll) {  
097 throw unsupportedOperationException();  
098 }  
099  
100 @Override  
101 public void clear() {  
102 throw unsupportedOperationException();  
103 }  
104  
105 @Override  
106 public E get(final int index) {  
107 return decorated().get(index);  
108 }  
109  
110 @Override  
111 public int indexOf(final Object object) {  
112 return decorated().indexOf(object);  
113 }  
114  
115 @Override  
116 public Iterator<E> iterator() {  
117 return UnmodifiableIterator.unmodifiableIterator(decorated().iterator());  
118 }  
119  
120 @Override  
121 public int lastIndexOf(final Object object) {  
122 return decorated().lastIndexOf(object);  
123 }  
124  
125 @Override  
126 public ListIterator<E> listIterator() {  
127 return new FixedSizeListIterator(decorated().listIterator(0));  
128 }  
129  
130 @Override  
131 public ListIterator<E> listIterator(final int index) {  
132 return new FixedSizeListIterator(decorated().listIterator(index));  
133 }  
134  
135 @Override  
136 public E remove(final int index) {  
137 throw unsupportedOperationException();  
138 }  
139  
140 @Override  
141 public boolean remove(final Object object) {  
142 throw unsupportedOperationException();  
143 }  
144  
145 /\*\*  
146 \* @since 4.4  
147 \*/  
148 @Override  
149 public boolean removeIf(Predicate<? super E> filter) {  
150 throw unsupportedOperationException();  
151 }  
152  
153 @Override  
154 public boolean removeAll(final Collection<?> coll) {  
155 throw unsupportedOperationException();  
156 }  
157  
158 @Override  
159 public boolean retainAll(final Collection<?> coll) {  
160 throw unsupportedOperationException();  
161 }  
162  
163 @Override  
164 public E set(final int index, final E object) {  
165 return decorated().set(index, object);  
166 }  
167  
168 @Override  
169 public List<E> subList(final int fromIndex, final int toIndex) {  
170 final List<E> sub = decorated().subList(fromIndex, toIndex);  
171 return new FixedSizeList<>(sub);  
172 }  
173  
174 /\*\*  
175 \* List iterator that only permits changes via set()  
176 \*/  
177 private class FixedSizeListIterator extends AbstractListIteratorDecorator<E> {  
178 protected FixedSizeListIterator(final ListIterator<E> iterator) {  
179 super(iterator);  
180 }  
181 @Override  
182 public void remove() {  
183 throw unsupportedOperationException();  
184 }  
185 @Override  
186 public void add(final Object object) {  
187 throw unsupportedOperationException();  
188 }  
189 }  
190  
191 @Override  
192 public boolean isFull() {  
193 return true;  
194 }  
195  
196 @Override  
197 public int maxSize() {  
198 return size();  
199 }  
200  
201 private static UnsupportedOperationException unsupportedOperationException() {  
202 return new UnsupportedOperationException("List is fixed size");  
203 }  
204  
205}