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.ArrayList;  
020import java.util.Collection;  
021import java.util.Collections;  
022import java.util.List;  
023  
024/\*\*  
025 \* Decorates another <code>List</code> to make it seamlessly grow when  
026 \* indices larger than the list size are used on add and set,  
027 \* avoiding most IndexOutOfBoundsExceptions.  
028 \* <p>  
029 \* This class avoids errors by growing when a set or add method would  
030 \* normally throw an IndexOutOfBoundsException.  
031 \* Note that IndexOutOfBoundsException IS returned for invalid negative indices.  
032 \* </p>  
033 \* <p>  
034 \* Trying to set or add to an index larger than the size will cause the list  
035 \* to grow (using <code>null</code> elements). Clearly, care must be taken  
036 \* not to use excessively large indices, as the internal list will grow to  
037 \* match.  
038 \* </p>  
039 \* <p>  
040 \* Trying to use any method other than add or set with an invalid index will  
041 \* call the underlying list and probably result in an IndexOutOfBoundsException.  
042 \* </p>  
043 \* <p>  
044 \* Take care when using this list with <code>null</code> values, as  
045 \* <code>null</code> is the value added when growing the list.  
046 \* </p>  
047 \* <p>  
048 \* All sub-lists will access the underlying list directly, and will throw  
049 \* IndexOutOfBoundsExceptions.  
050 \* </p>  
051 \* <p>  
052 \* This class differs from {@link LazyList} because here growth occurs on  
053 \* set and add, where <code>LazyList</code> grows on get. However, they  
054 \* can be used together by decorating twice.  
055 \* </p>  
056 \*  
057 \* @see LazyList  
058 \* @since 3.2  
059 \*/  
060public class GrowthList<E> extends AbstractSerializableListDecorator<E> {  
061  
062 /\*\* Serialization version \*/  
063 private static final long serialVersionUID = -3620001881672L;  
064  
065 /\*\*  
066 \* Factory method to create a growth list.  
067 \*  
068 \* @param <E> the type of the elements in the list  
069 \* @param list the list to decorate, must not be null  
070 \* @return a new growth list  
071 \* @throws NullPointerException if list is null  
072 \* @since 4.0  
073 \*/  
074 public static <E> GrowthList<E> growthList(final List<E> list) {  
075 return new GrowthList<>(list);  
076 }  
077  
078 //-----------------------------------------------------------------------  
079 /\*\*  
080 \* Constructor that uses an ArrayList internally.  
081 \*/  
082 public GrowthList() {  
083 super(new ArrayList<E>());  
084 }  
085  
086 /\*\*  
087 \* Constructor that uses an ArrayList internally.  
088 \*  
089 \* @param initialSize the initial size of the ArrayList  
090 \* @throws IllegalArgumentException if initial size is invalid  
091 \*/  
092 public GrowthList(final int initialSize) {  
093 super(new ArrayList<E>(initialSize));  
094 }  
095  
096 /\*\*  
097 \* Constructor that wraps (not copies).  
098 \*  
099 \* @param list the list to decorate, must not be null  
100 \* @throws NullPointerException if list is null  
101 \*/  
102 protected GrowthList(final List<E> list) {  
103 super(list);  
104 }  
105  
106 //-----------------------------------------------------------------------  
107 /\*\*  
108 \* Decorate the add method to perform the growth behaviour.  
109 \* <p>  
110 \* If the requested index is greater than the current size, the list will  
111 \* grow to the new size. Indices between the old size and the requested  
112 \* size will be filled with <code>null</code>.  
113 \* <p>  
114 \* If the index is less than the current size, the value will be added to  
115 \* the underlying list directly.  
116 \* If the index is less than zero, the underlying list is called, which  
117 \* will probably throw an IndexOutOfBoundsException.  
118 \*  
119 \* @param index the index to add at  
120 \* @param element the object to add at the specified index  
121 \* @throws UnsupportedOperationException if the underlying list doesn't implement set  
122 \* @throws ClassCastException if the underlying list rejects the element  
123 \* @throws IllegalArgumentException if the underlying list rejects the element  
124 \*/  
125 @Override  
126 public void add(final int index, final E element) {  
127 final int size = decorated().size();  
128 if (index > size) {  
129 decorated().addAll(Collections.<E>nCopies(index - size, null));  
130 }  
131 decorated().add(index, element);  
132 }  
133  
134 //-----------------------------------------------------------------------  
135 /\*\*  
136 \* Decorate the addAll method to perform the growth behaviour.  
137 \* <p>  
138 \* If the requested index is greater than the current size, the list will  
139 \* grow to the new size. Indices between the old size and the requested  
140 \* size will be filled with <code>null</code>.  
141 \* <p>  
142 \* If the index is less than the current size, the values will be added to  
143 \* the underlying list directly.  
144 \* If the index is less than zero, the underlying list is called, which  
145 \* will probably throw an IndexOutOfBoundsException.  
146 \*  
147 \* @param index the index to add at  
148 \* @param coll the collection to add at the specified index  
149 \* @return true if the list changed  
150 \* @throws UnsupportedOperationException if the underlying list doesn't implement set  
151 \* @throws ClassCastException if the underlying list rejects the element  
152 \* @throws IllegalArgumentException if the underlying list rejects the element  
153 \*/  
154 @Override  
155 public boolean addAll(final int index, final Collection<? extends E> coll) {  
156 final int size = decorated().size();  
157 boolean result = false;  
158 if (index > size) {  
159 decorated().addAll(Collections.<E>nCopies(index - size, null));  
160 result = true;  
161 }  
162 return decorated().addAll(index, coll) || result;  
163 }  
164  
165 //-----------------------------------------------------------------------  
166 /\*\*  
167 \* Decorate the set method to perform the growth behaviour.  
168 \* <p>  
169 \* If the requested index is greater than the current size, the list will  
170 \* grow to the new size. Indices between the old size and the requested  
171 \* size will be filled with <code>null</code>.  
172 \* <p>  
173 \* If the index is less than the current size, the value will be set onto  
174 \* the underlying list directly.  
175 \* If the index is less than zero, the underlying list is called, which  
176 \* will probably throw an IndexOutOfBoundsException.  
177 \*  
178 \* @param index the index to set  
179 \* @param element the object to set at the specified index  
180 \* @return the object previously at that index  
181 \* @throws UnsupportedOperationException if the underlying list doesn't implement set  
182 \* @throws ClassCastException if the underlying list rejects the element  
183 \* @throws IllegalArgumentException if the underlying list rejects the element  
184 \*/  
185 @Override  
186 public E set(final int index, final E element) {  
187 final int size = decorated().size();  
188 if (index >= size) {  
189 decorated().addAll(Collections.<E>nCopies(index - size + 1, null));  
190 }  
191 return decorated().set(index, element);  
192 }  
193  
194}