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.List;  
020import java.util.Objects;  
021  
022import org.apache.commons.collections4.Factory;  
023import org.apache.commons.collections4.Transformer;  
024  
025/\*\*  
026 \* Decorates another <code>List</code> to create objects in the list on demand.  
027 \* <p>  
028 \* When the {@link #get(int)} method is called with an index greater than  
029 \* the size of the list, the list will automatically grow in size and return  
030 \* a new object from the specified factory or transformer. The gaps will be  
031 \* filled by null. If a get method call encounters a null, it will be replaced  
032 \* with a new object from the factory. Thus this list is unsuitable for  
033 \* storing null objects.  
034 \* </p>  
035 \* <p>  
036 \* For instance:  
037 \* </p>  
038 \*  
039 \* <pre>  
040 \* Factory<Date> factory = new Factory<Date>() {  
041 \* public Date create() {  
042 \* return new Date();  
043 \* }  
044 \* }  
045 \* List<Date> lazy = LazyList.decorate(new ArrayList<Date>(), factory);  
046 \* Date date = lazy.get(3);  
047 \* </pre>  
048 \*  
049 \* <p>  
050 \* After the above code is executed, <code>date</code> will contain  
051 \* a new <code>Date</code> instance. Furthermore, that <code>Date</code>  
052 \* instance is the fourth element in the list. The first, second,  
053 \* and third element are all set to <code>null</code>.  
054 \* </p>  
055 \* <p>  
056 \* This class differs from {@link GrowthList} because here growth occurs on  
057 \* get, where <code>GrowthList</code> grows on set and add. However, they  
058 \* could easily be used together by decorating twice.  
059 \* </p>  
060 \* <p>  
061 \* This class is Serializable from Commons Collections 3.1.  
062 \* </p>  
063 \*  
064 \* @see GrowthList  
065 \* @since 3.0  
066 \*/  
067public class LazyList<E> extends AbstractSerializableListDecorator<E> {  
068  
069 /\*\* Serialization version \*/  
070 private static final long serialVersionUID = -3677737457567429713L;  
071  
072 /\*\* The factory to use to lazily instantiate the objects \*/  
073 private final Factory<? extends E> factory;  
074  
075 /\*\* The transformer to use to lazily instantiate the objects \*/  
076 private final Transformer<Integer, ? extends E> transformer;  
077  
078 /\*\*  
079 \* Factory method to create a lazily instantiating list.  
080 \*  
081 \* @param <E> the type of the elements in the list  
082 \* @param list the list to decorate, must not be null  
083 \* @param factory the factory to use for creation, must not be null  
084 \* @return a new lazy list  
085 \* @throws NullPointerException if list or factory is null  
086 \* @since 4.0  
087 \*/  
088 public static <E> LazyList<E> lazyList(final List<E> list, final Factory<? extends E> factory) {  
089 return new LazyList<>(list, factory);  
090 }  
091  
092 /\*\*  
093 \* Transformer method to create a lazily instantiating list.  
094 \*  
095 \* @param <E> the type of the elements in the list  
096 \* @param list the list to decorate, must not be null  
097 \* @param transformer the transformer to use for creation, must not be null  
098 \* @return a new lazy list  
099 \* @throws NullPointerException if list or transformer is null  
100 \* @since 4.4  
101 \*/  
102 public static <E> LazyList<E> lazyList(final List<E> list, final Transformer<Integer, ? extends E> transformer) {  
103 return new LazyList<>(list, transformer);  
104 }  
105  
106 //-----------------------------------------------------------------------  
107 /\*\*  
108 \* Constructor that wraps (not copies).  
109 \*  
110 \* @param list the list to decorate, must not be null  
111 \* @param factory the factory to use for creation, must not be null  
112 \* @throws NullPointerException if list or factory is null  
113 \*/  
114 protected LazyList(final List<E> list, final Factory<? extends E> factory) {  
115 super(list);  
116 this.factory = Objects.requireNonNull(factory);  
117 this.transformer = null;  
118 }  
119  
120 /\*\*  
121 \* Constructor that wraps (not copies).  
122 \*  
123 \* @param list the list to decorate, must not be null  
124 \* @param transformer the transformer to use for creation, must not be null  
125 \* @throws NullPointerException if list or transformer is null  
126 \*/  
127 protected LazyList(final List<E> list, final Transformer<Integer, ? extends E> transformer) {  
128 super(list);  
129 this.factory = null;  
130 this.transformer = Objects.requireNonNull(transformer);  
131 }  
132  
133 //-----------------------------------------------------------------------  
134 /\*\*  
135 \* Decorate the get method to perform the lazy behaviour.  
136 \* <p>  
137 \* If the requested index is greater than the current size, the list will  
138 \* grow to the new size and a new object will be returned from the factory  
139 \* or transformer. Indexes in-between the old size and the requested size  
140 \* are left with a placeholder that is replaced with a factory or  
141 \* transformer object when requested.  
142 \*  
143 \* @param index the index to retrieve  
144 \* @return the element at the given index  
145 \*/  
146 @Override  
147 public E get(final int index) {  
148 final int size = decorated().size();  
149 if (index < size) {  
150 // within bounds, get the object  
151 E object = decorated().get(index);  
152 if (object == null) {  
153 // item is a place holder, create new one, set and return  
154 object = element(index);  
155 decorated().set(index, object);  
156 return object;  
157 }  
158 // good and ready to go  
159 return object;  
160 }  
161 // we have to grow the list  
162 for (int i = size; i < index; i++) {  
163 decorated().add(null);  
164 }  
165 // create our last object, set and return  
166 final E object = element(index);  
167 decorated().add(object);  
168 return object;  
169 }  
170  
171 @Override  
172 public List<E> subList(final int fromIndex, final int toIndex) {  
173 final List<E> sub = decorated().subList(fromIndex, toIndex);  
174 if (factory != null) {  
175 return new LazyList<>(sub, factory);  
176 } else if (transformer != null) {  
177 return new LazyList<>(sub, transformer);  
178 } else {  
179 throw new IllegalStateException("Factory and Transformer are both null!");  
180 }  
181 }  
182  
183 private E element(final int index) {  
184 if (factory != null) {  
185 return factory.create();  
186 } else if (transformer != null) {  
187 return transformer.transform(index);  
188 } else {  
189 throw new IllegalStateException("Factory and Transformer are both null!");  
190 }  
191 }  
192  
193}