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.collection.AbstractCollectionDecorator;  
024  
025/\*\*  
026 \* Decorates another {@link List} to provide additional behaviour.  
027 \* <p>  
028 \* Methods are forwarded directly to the decorated list.  
029 \* </p>  
030 \*  
031 \* @param <E> the type of the elements in the list  
032 \* @since 3.0  
033 \*/  
034public abstract class AbstractListDecorator<E> extends AbstractCollectionDecorator<E>  
035 implements List<E> {  
036  
037 /\*\* Serialization version--necessary in an abstract class? \*/  
038 private static final long serialVersionUID = 4500739654952315623L;  
039  
040 /\*\*  
041 \* Constructor only used in deserialization, do not use otherwise.  
042 \* @since 3.1  
043 \*/  
044 protected AbstractListDecorator() {  
045 super();  
046 }  
047  
048 /\*\*  
049 \* Constructor that wraps (not copies).  
050 \*  
051 \* @param list the list to decorate, must not be null  
052 \* @throws NullPointerException if list is null  
053 \*/  
054 protected AbstractListDecorator(final List<E> list) {  
055 super(list);  
056 }  
057  
058 /\*\*  
059 \* Gets the list being decorated.  
060 \*  
061 \* @return the decorated list  
062 \*/  
063 @Override  
064 protected List<E> decorated() {  
065 return (List<E>) super.decorated();  
066 }  
067  
068 @Override  
069 public boolean equals(final Object object) {  
070 return object == this || decorated().equals(object);  
071 }  
072  
073 @Override  
074 public int hashCode() {  
075 return decorated().hashCode();  
076 }  
077  
078 //-----------------------------------------------------------------------  
079  
080 @Override  
081 public void add(final int index, final E object) {  
082 decorated().add(index, object);  
083 }  
084  
085 @Override  
086 public boolean addAll(final int index, final Collection<? extends E> coll) {  
087 return decorated().addAll(index, coll);  
088 }  
089  
090 @Override  
091 public E get(final int index) {  
092 return decorated().get(index);  
093 }  
094  
095 @Override  
096 public int indexOf(final Object object) {  
097 return decorated().indexOf(object);  
098 }  
099  
100 @Override  
101 public int lastIndexOf(final Object object) {  
102 return decorated().lastIndexOf(object);  
103 }  
104  
105 @Override  
106 public ListIterator<E> listIterator() {  
107 return decorated().listIterator();  
108 }  
109  
110 @Override  
111 public ListIterator<E> listIterator(final int index) {  
112 return decorated().listIterator(index);  
113 }  
114  
115 @Override  
116 public E remove(final int index) {  
117 return decorated().remove(index);  
118 }  
119  
120 @Override  
121 public E set(final int index, final E object) {  
122 return decorated().set(index, object);  
123 }  
124  
125 @Override  
126 public List<E> subList(final int fromIndex, final int toIndex) {  
127 return decorated().subList(fromIndex, toIndex);  
128 }  
129  
130}