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;  
018  
019import java.util.ArrayList;  
020import java.util.EmptyStackException;  
021  
022/\*\*  
023 \* An implementation of the {@link java.util.Stack} API that is based on an  
024 \* <code>ArrayList</code> instead of a <code>Vector</code>, so it is not  
025 \* synchronized to protect against multi-threaded access. The implementation  
026 \* is therefore operates faster in environments where you do not need to  
027 \* worry about multiple thread contention.  
028 \* <p>  
029 \* The removal order of an <code>ArrayStack</code> is based on insertion  
030 \* order: The most recently added element is removed first. The iteration  
031 \* order is <i>not</i> the same as the removal order. The iterator returns  
032 \* elements from the bottom up.  
033 \* </p>  
034 \* <p>  
035 \* Unlike <code>Stack</code>, <code>ArrayStack</code> accepts null entries.  
036 \* <p>  
037 \* <b>Note:</b> From version 4.0 onwards, this class does not implement the  
038 \* removed {@code Buffer} interface anymore.  
039 \* </p>  
040 \*  
041 \* @param <E> the type of elements in this list  
042 \* @see java.util.Stack  
043 \* @since 1.0  
044 \* @deprecated use {@link java.util.ArrayDeque} instead (available from Java 1.6)  
045 \*/  
046@Deprecated  
047public class ArrayStack<E> extends ArrayList<E> {  
048  
049 /\*\* Ensure serialization compatibility \*/  
050 private static final long serialVersionUID = 2130079159931574599L;  
051  
052 /\*\*  
053 \* Constructs a new empty <code>ArrayStack</code>. The initial size  
054 \* is controlled by <code>ArrayList</code> and is currently 10.  
055 \*/  
056 public ArrayStack() {  
057 super();  
058 }  
059  
060 /\*\*  
061 \* Constructs a new empty <code>ArrayStack</code> with an initial size.  
062 \*  
063 \* @param initialSize the initial size to use  
064 \* @throws IllegalArgumentException if the specified initial size  
065 \* is negative  
066 \*/  
067 public ArrayStack(final int initialSize) {  
068 super(initialSize);  
069 }  
070  
071 /\*\*  
072 \* Return <code>true</code> if this stack is currently empty.  
073 \* <p>  
074 \* This method exists for compatibility with <code>java.util.Stack</code>.  
075 \* New users of this class should use <code>isEmpty</code> instead.  
076 \*  
077 \* @return true if the stack is currently empty  
078 \*/  
079 public boolean empty() {  
080 return isEmpty();  
081 }  
082  
083 /\*\*  
084 \* Returns the top item off of this stack without removing it.  
085 \*  
086 \* @return the top item on the stack  
087 \* @throws EmptyStackException if the stack is empty  
088 \*/  
089 public E peek() throws EmptyStackException {  
090 final int n = size();  
091 if (n <= 0) {  
092 throw new EmptyStackException();  
093 }  
094 return get(n - 1);  
095 }  
096  
097 /\*\*  
098 \* Returns the n'th item down (zero-relative) from the top of this  
099 \* stack without removing it.  
100 \*  
101 \* @param n the number of items down to go  
102 \* @return the n'th item on the stack, zero relative  
103 \* @throws EmptyStackException if there are not enough items on the  
104 \* stack to satisfy this request  
105 \*/  
106 public E peek(final int n) throws EmptyStackException {  
107 final int m = (size() - n) - 1;  
108 if (m < 0) {  
109 throw new EmptyStackException();  
110 }  
111 return get(m);  
112 }  
113  
114 /\*\*  
115 \* Pops the top item off of this stack and return it.  
116 \*  
117 \* @return the top item on the stack  
118 \* @throws EmptyStackException if the stack is empty  
119 \*/  
120 public E pop() throws EmptyStackException {  
121 final int n = size();  
122 if (n <= 0) {  
123 throw new EmptyStackException();  
124 }  
125 return remove(n - 1);  
126 }  
127  
128 /\*\*  
129 \* Pushes a new item onto the top of this stack. The pushed item is also  
130 \* returned. This is equivalent to calling <code>add</code>.  
131 \*  
132 \* @param item the item to be added  
133 \* @return the item just pushed  
134 \*/  
135 public E push(final E item) {  
136 add(item);  
137 return item;  
138 }  
139  
140 /\*\*  
141 \* Returns the one-based position of the distance from the top that the  
142 \* specified object exists on this stack, where the top-most element is  
143 \* considered to be at distance <code>1</code>. If the object is not  
144 \* present on the stack, return <code>-1</code> instead. The  
145 \* <code>equals()</code> method is used to compare to the items  
146 \* in this stack.  
147 \*  
148 \* @param object the object to be searched for  
149 \* @return the 1-based depth into the stack of the object, or -1 if not found  
150 \*/  
151 public int search(final Object object) {  
152 int i = size() - 1; // Current index  
153 int n = 1; // Current distance  
154 while (i >= 0) {  
155 final Object current = get(i);  
156 if ((object == null && current == null) ||  
157 (object != null && object.equals(current))) {  
158 return n;  
159 }  
160 i--;  
161 n++;  
162 }  
163 return -1;  
164 }  
165  
166}