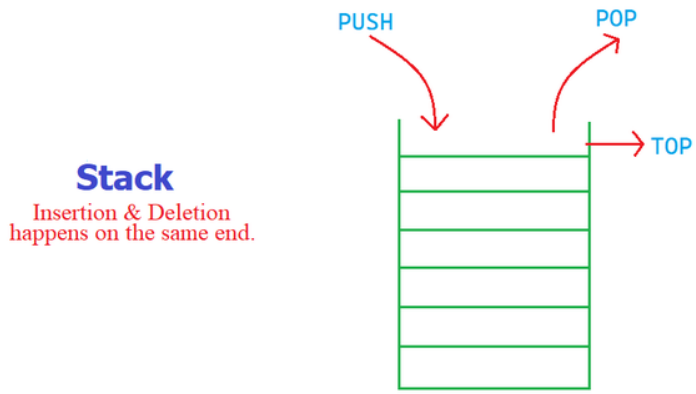
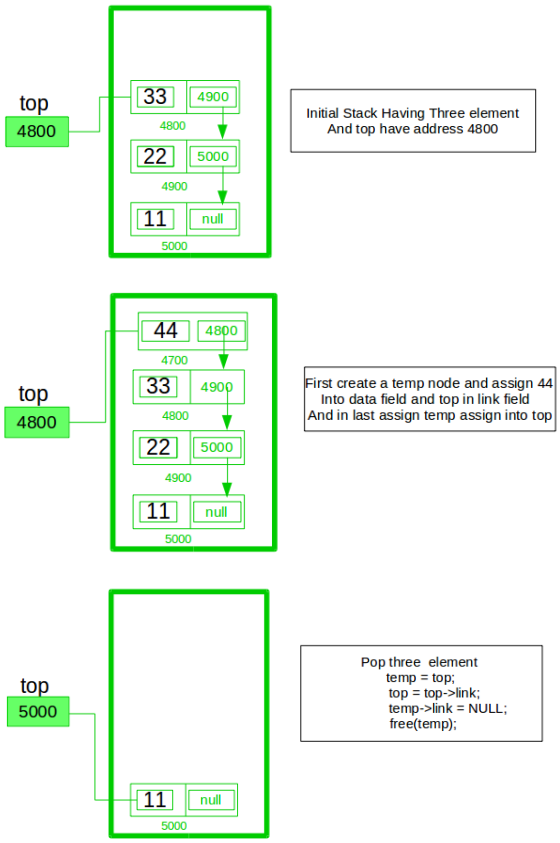
Bab Stack

- Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).
- Mainly the following four basic operations are performed in the stack:
    - **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
    - **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
    - **Peek or Top:** Returns the top element of the stack.
    - **isEmpty:** Returns true if the stack is empty, else false.



Implement Stack using singly linked list

To implement a stack using singly linked list concept , all the singly linked list operations are performed based on Stack operations LIFO(last in first out) and with the help of that knowledge we are going to implement a stack using single linked list. Using singly linked lists , we implement stack by storing the information in the form of nodes and we need to follow the stack rules and implement using singly linked list nodes . So we need to follow a simple rule in the implementation of a stack which is last in first out and all the operations can be performed with the help of a top variable .Let us learn how to perform **Pop , Push , Peek ,Display** operations in the following article .



A stack can be easily implemented using the linked list. In stack Implementation, a stack contains a top pointer. which is “head” of the stack where pushing and popping items happens at the head of the list. First node have null in link field and second node link have first node address in link field and so on and last node address in “top” pointer.
The main advantage of using linked list over an arrays is that it is possible to implement a stack that can shrink or grow as much as needed. In using array will put a restriction to the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocate. so overflow is not possible.

Stack Operations:

1. **push() :** Insert a new element into stack i.e just inserting a new element at the beginning of the linked list.
2. **pop() :** Return top element of the Stack i.e simply deleting the first element from the linked list.
3. **peek():** Return the top element.
4. **display():** Print all elements in Stack.

```
// Java program to Implement a stack
// using singly linked list
// import package
import static java.lang.System.exit;

// Create Stack Using Linked list
```

```
class StackUsingLinkedlist {

    // A linked list node
    private class Node {

        int data; // integer data
        Node link; // reference variable Node type
    }
    // create global top reference variable global
    Node top;
    // Constructor
    StackUsingLinkedlist()
    {
        this.top = null;
    }

    // Utility function to add an element x in the stack
    public void push(int x) // insert at the beginning
    {
        // create new node temp and allocate memory
        Node temp = new Node();

        // check if stack (heap) is full. Then inserting an
        // element would lead to stack overflow
        if (temp == null) {
            System.out.print("\nHeap Overflow");
            return;
        }

        // initialize data into temp data field
        temp.data = x;

        // put top reference into temp link
        temp.link = top;

        // update top reference
        top = temp;
    }

    // Utility function to check if the stack is empty or not
    public boolean isEmpty()
    {
        return top == null;
    }

    // Utility function to return top element in a stack
    public int peek()
    {
```

```
                // check for empty stack
                if (!isEmpty()) {
                        return top.data;
                }
                else {
                        System.out.println("Stack is empty");
                        return -1;
                }
        }

        // Utility function to pop top element from the stack
        public void pop() // remove at the beginning
        {
                // check for stack underflow
                if (top == null) {
                        System.out.print("\nStack Underflow");
                        return;
                }

                // update the top pointer to point to the next node
                top = (top).link;
        }

        public void display()
        {
                // check for stack underflow
                if (top == null) {
                        System.out.printf("\nStack Underflow");
                        exit(1);
                }
                else {
                        Node temp = top;
                        while (temp != null) {

                                // print node data
                                System.out.printf("%d->", temp.data);

                                // assign temp link to temp
                                temp = temp.link;
                        }
                }
        }
}
// main class
public class GFG {
        public static void main(String[] args)
        {
                // create Object of Implementing class
```

```
        StackUsingLinkedlist obj = new StackUsingLinkedlist();
        // insert Stack value
        obj.push(11);
        obj.push(22);
        obj.push(33);
        obj.push(44);

        // print Stack elements
        obj.display();

        // print Top element of Stack
        System.out.printf("\nTop element is %d\n", obj.peek());

        // Delete top element of Stack
        obj.pop();
        obj.pop();

        // print Stack elements
        obj.display();

        // print Top element of Stack
        System.out.printf("\nTop element is %d\n", obj.peek());
    }
}
```

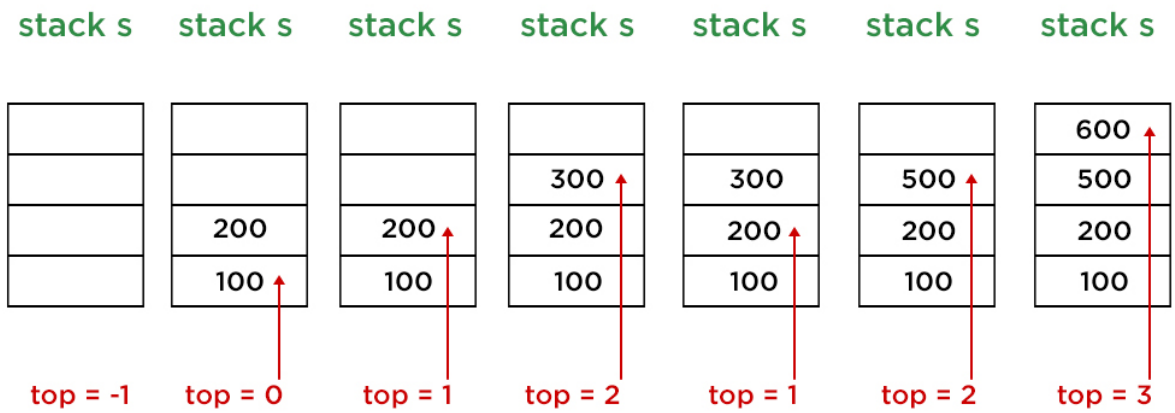
Implement Stack in Java Using Array and Generics

Stack is a linear Data Structure that is based on the LIFO concept (last in first out). Instead of only an Integer Stack, Stack can be of String, Character, or even Float type. There are 4 primary operations in the stack as follows:

1. *push() Method* adds element x to the stack.
2. *pop() Method* removes *the* last element of *the* stack.
3. *top()* Method returns the last element of the stack.
4. *empty() Method* returns whether *the* stack is empty or not.

Note: *Time Complexity is of order 1 for all operations of the stack*

Illustration:



Stack 1

```
let s = empty stack of Integer type with size 4
```

Stack 2

```
push (100) : top = top + 1 and s[top] = 100
```

Stack 3

```
push (200) : top = top + 1 and s[top] = 200
```

Stack 4

```
 push (300) : top = top + 1 and s[top] = 300
```

Stack 5

```
pop ( )  : top = top - 1
```

Stack 6

```
push (500) : top = top + 1 and s[top] = 500
```

Stack 7

```
push (600) : top = top + 1 and s[top] = 600
```

Note:

push (700) : top +1 == size of stack : Stack Overflow !

// Since top = 3 and size of stack = 4, no more elements can be pushed

Implementation:

```
// Java Program to Implement Stack in Java Using Array and
// Generics

// Importing input output classes
import java.io.*;
```

```
// Importing all utility classes
import java.util.*;

// user defined class for generic stack
class stack<T> {

	// Empty array list
	ArrayList<T> A;

	// Default value of top variable when stack is empty
	int top = -1;

	// Variable to store size of array
	int size;

	// Constructor of this class
	// To initialize stack
	stack(int size)
	{
		// Storing the value of size into global variable
		this.size = size;

		// Creating array of Size = size
		this.A = new ArrayList<T>(size);
	}

	// Method 1
	// To push generic element into stack
	void push(T X)
	{
		// Checking if array is full
		if (top + 1 == size) {

			// Display message when array is full
			System.out.println("Stack Overflow");
		}
		else {

			// Increment top to go to next position
			top = top + 1;

			// Over-writing existing element
			if (A.size() > top)
				A.set(top, X);

			else

				// Creating new element
```

```
                A.add(X);
            }
        }
        // Method 2
        // To return topmost element of stack
        T top()
        {
            // If stack is empty
            if (top == -1) {

                // Display message when there are no elements in
                // the stack
                System.out.println("Stack Underflow");

                return null;
            }

            // else elements are present so
            // return the topmost element
            else
                return A.get(top);
        }

        // Method 3
        // To delete last element of stack
        void pop()
        {
            // If stack is empty
            if (top == -1) {

                // Display message when there are no elements in
                // the stack
                System.out.println("Stack Underflow");
            }

            else

                // Delete the last element
                // by decrementing the top
                top--;
        }

        // Method 4
        // To check if stack is empty or not
        boolean empty() { return top == -1; }

        // Method 5
        // To print the stack
```

```
        // @Override
        public String toString()
        {

                String Ans = "";

                for (int i = 0; i < top; i++) {
                        Ans += String.valueOf(A.get(i)) + "->";
                }

                Ans += String.valueOf(A.get(top));

                return Ans;
        }
}
// Main Class
public class GFG {

        // main driver method
        public static void main(String[] args)
        {

                // Integer Stack

                // Creating an object of Stack class
                // Declaring objects of Integer type
                stack<Integer> s1 = new stack<>(3);

                // Pushing elements to integer stack - s1

                // Element 1 - 10
                s1.push(10);
                // Element 2 - 20
                s1.push(20);
                // Element 3 - 30
                s1.push(30);

                // Print the stack elements after pushing the
                // elements
                System.out.println(
                        "s1 after pushing 10, 20 and 30 :\n" + s1);

                // Now, pop from stack s1
                s1.pop();

                // Print the stack elements after poping few
                // element/s
                System.out.println("s1 after pop :\n" + s1);
```

```
        // String Stack

        // Creating an object of Stack class
        // Declaring objects of Integer type
        stack<String> s2 = new stack<>(3);

        // Pushing elements to string stack - s2

        // Element 1 - hello
        s2.push("hello");
        // Element 2 - world
        s2.push("world");
        // Element 3 - java
        s2.push("java");

        // Print string stack after pushing above string
        // elements
        System.out.println(
                "\ns2 after pushing 3 elements :\n" + s2);

        System.out.println(
                "s2 after pushing 4th element :");

        // Pushing another element to above stack

        // Element 4 - GFG
        s2.push("GFG");

        // Float stack

        // Creating an object of Stack class
        // Declaring objects of Integer type
        stack<Float> s3 = new stack<>(2);

        // Pushing elements to float stack - s3

        // Element 1 - 100.0
        s3.push(100.0f);
        // Element 2 - 200.0
        s3.push(200.0f);

        // Print string stack after pushing above float
        // elements
        System.out.println(
                "\ns3 after pushing 2 elements :\n" + s3);

        // Print and display top element of stack s3
```

```
                System.out.println("top element of s3:\n"
                                                + s3.top());
        }
}
```

Stack Basics: last-in, first-out behavior

```
import java.util.Stack;

public class MainClass {
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push("A");
    s.push("B");
    s.push("C");

    System.out.println(s);
  }
}
```

Adding Elements: To add an element to a stack, call the push() method

This adds the element to the top of the stack or to the end of the vector.

```
import java.util.Stack;

public class MainClass {
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push("A");
    s.push("B");
    s.push("C");

    System.out.println(s);
  }
```

```
}
```

Removing Elements: To remove an element from the stack, the pop() method

Taking the element at the top of the stack, removes it, and returns it.

If the stack is empty when called, you will get the runtime EmptyStackException thrown.

```
import java.util.Stack;

public class MainClass {
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push("A");
    s.push("B");
    s.push("C");

    System.out.println(s.pop());

  }
}
```

If the size of the stack is zero, true is returned; otherwise, false is returned

```
import java.util.Stack;

public class MainClass {
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push("A");
    s.push("B");
    s.push("C");

    System.out.println(s.pop());
```

```
    System.out.println(s.empty());
  }
}
```

Checking the Top: To get the element without removing: using the peek() method

You'll get an EmptyStackException thrown if the stack is empty.

```
import java.util.Stack;

public class MainClass {
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push("A");
    s.push("B");
    s.push("C");

    System.out.println("Next: " + s.peek());
  }
}
```

To find out if an element is on the stack: the search() method

The element at the top of the stack is at position 1.

Position 2 is next, then 3, and so on.

If the requested object is not found on the stack, -1 is returned.

```
import java.util.Stack;

public class MainClass {
  public static void main (String args[]) {
    Stack s = new Stack();
    s.push("A");
    s.push("B");
```

```
    s.push("C");

    System.out.println("Next: " + s.peek());
    s.push("D");

    System.out.println(s.pop());
    s.push("E");
    s.push("F");

    int count = s.search("E");
    while (count != -1 && count > 1) {
      s.pop();
      count--;
    }
    System.out.println(s);
  }
}
```

Demonstrate the generic Stack class.

```
import java.util.EmptyStackException;
import java.util.Stack;

class StackDemo {
  static void showpush(Stack<Integer> st, int a) {
    st.push(a);
    System.out.println("push(" + a + ")");
    System.out.println("stack: " + st);
  }

  static void showpop(Stack<Integer> st) {
    System.out.print("pop -> ");
    Integer a = st.pop();
    System.out.println(a);
```

```
    System.out.println("stack: " + st);
  }

  public static void main(String args[]) {
    Stack<Integer> st = new Stack<Integer>();

    System.out.println("stack: " + st);
    showpush(st, 42);
    showpush(st, 66);
    showpush(st, 99);
    showpop(st);
    showpop(st);
    showpop(st);

    try {
      showpop(st);
    } catch (EmptyStackException e) {
      System.out.println("empty stack");
    }
  }
}
```

A faster, smaller stack implementation.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
```

```
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

import java.util.ArrayList;
import java.util.Collection;
import java.util.EmptyStackException;

/**
 * A faster, smaller stack implementation. ArrayListStack is final and unsynchroni
zed (the JDK's
 * methods are synchronized). In addition you can set the initial capacity if you
want via the
 * ArrayListStack(int) constructor.
 *
 * @author Jonathan Locke
 * @param <T>
 */
public final class ArrayListStack<T> extends ArrayList<T>
{
  private static final long serialVersionUID = 1L;

  /**
   * Construct.
   *
   * @param initialCapacity
   *            Initial capacity of the stack
   */
  public ArrayListStack(final int initialCapacity)
  {
    super(initialCapacity);
  }

  /**
   * Construct.
```

```
 */
public ArrayListStack()
{
  this(10);
}

/**
 * Construct.
 *
 * @param collection
 *            The collection to add
 */
public ArrayListStack(final Collection<T> collection)
{
  super(collection);
}

/**
 * Pushes an item onto the top of this stack.
 *
 * @param item
 *            the item to be pushed onto this stack.
 */
public final void push(final T item)
{
  add(item);
}

/**
 * Removes the object at the top of this stack and returns that object.
 *
 * @return The object at the top of this stack
 * @exception EmptyStackException
 *                If this stack is empty.
 */
public final T pop()
```

```
  {
    final T top = peek();
    remove(size() - 1);
    return top;
  }

  /**
   * Looks at the object at the top of this stack without removing it.
   *
   * @return The object at the top of this stack
   * @exception EmptyStackException
   *                If this stack is empty.
   */
  public final T peek()
  {
    int size = size();
    if (size == 0)
    {
      throw new EmptyStackException();
    }
    return get(size - 1);
  }

  /**
   * Tests if this stack is empty.
   *
   * @return <code>true</code> if and only if this stack contains no items; <code>
false</code>
   *         otherwise.
   */
  public final boolean empty()
  {
    return size() == 0;
  }

  /**
```

```
   * Returns the 1-based position where an object is on this stack. If the object
<tt>o</tt>
   * occurs as an item in this stack, this method returns the distance from the to
p of the stack
   * of the occurrence nearest the top of the stack; the topmost item on the stack
is considered
   * to be at distance <tt>1</tt>. The <tt>equals</tt> method is used to compare <
tt>o</tt>
   * to the items in this stack.
   *
   * @param o
   *            the desired object.
   * @return the 1-based position from the top of the stack where the object is lo
cated; the
   *         return value <code>-1</code> indicates that the object is not on the
stack.
   */
  public final int search(final T o)
  {
    int i = lastIndexOf(o);
    if (i >= 0)
    {
      return size() - i;
    }
    return -1;
  }
}
```

A simple integer based stack.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
```

```
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */



/**
 * A simple integer based stack.
 *
 * moved to org.apache.xerces.util by neilg to support the
 * XPathMatcher.
 * @author  Andy Clark, IBM
 *
 * @version $Id: IntStack.java 447241 2006-09-18 05:12:57Z mrglavas $
 */
public final class IntStack {

    //
    // Data
    //

    /** Stack depth. */
    private int fDepth;

    /** Stack data. */
    private int[] fData;

    //
    // Public methods
    //
```

```
    /** Returns the size of the stack. */
    public int size() {
        return fDepth;
    }

    /** Pushes a value onto the stack. */
    public void push(int value) {
        ensureCapacity(fDepth + 1);
        fData[fDepth++] = value;
    }

    /** Peeks at the top of the stack. */
    public int peek() {
        return fData[fDepth - 1];
    }

    /** Returns the element at the specified depth in the stack. */
    public int elementAt(int depth) {
        return fData[depth];
    }

    /** Pops a value off of the stack. */
    public int pop() {
        return fData[--fDepth];
    }

    /** Clears the stack. */
    public void clear() {
        fDepth = 0;
    }

    // debugging

    /** Prints the stack. */
    public void print() {
        System.out.print('(');
```

```
        System.out.print(fDepth);
        System.out.print(") {");
        for (int i = 0; i < fDepth; i++) {
            if (i == 3) {
                System.out.print(" ...");
                break;
            }
            System.out.print(' ');
            System.out.print(fData[i]);
            if (i < fDepth - 1) {
                System.out.print(',');
            }
        }
        System.out.print(" }");
        System.out.println();
    }

    //
    // Private methods
    //

    /** Ensures capacity. */
    private void ensureCapacity(int size) {
        if (fData == null) {
            fData = new int[32];
        }
        else if (fData.length <= size) {
            int[] newdata = new int[fData.length * 2];
            System.arraycopy(fData, 0, newdata, 0, fData.length);
            fData = newdata;
        }
    }

} // class IntStack
```

Character Stack

```java
/*******************************************************************************
 * Copyright (c) 2008 xored software, Inc.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     xored software, Inc. - initial API and Implementation (Alex Panchenko)
 *******************************************************************************/


import java.util.EmptyStackException;

public class CharacterStack {

  private char[] buffer;
  private int size;

  public CharacterStack() {
    this(16);
  }

  /**
   * @param capacity
   */
  public CharacterStack(int capacity) {
    buffer = new char[capacity];
    size = 0;
  }

  /**
   * @return
   */
  public int size() {
```

```
    return size;
  }

  /**
   * @return
   */
  public char peek() {
    final int len = size;
    if (size == 0) {
      throw new EmptyStackException();
    }
    return buffer[len - 1];
  }

  /**
   * @return
   */
  public char pop() {
    int len = size;
    if (len == 0) {
      throw new EmptyStackException();
    }
    --len;
    final char result = buffer[len];
    size = len;
    return result;
  }

  /**
   * @param c
   */
  public void push(char c) {
    if (size == buffer.length) {
      char[] newBuffer = new char[buffer.length * 2 + 1];
      System.arraycopy(buffer, 0, newBuffer, 0, buffer.length);
      buffer = newBuffer;
```

```
    }
    buffer[size++] = c;
  }

}
```

Stack for Boolean

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the  "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
/*
 * $Id: BoolStack.java 468655 2006-10-28 07:12:06Z minchau $
 */

/**
 * Simple stack for boolean values.
 * @xsl.usage internal
 */
public final class BoolStack implements Cloneable
{
```

```java
/** Array of boolean values          */
private boolean m_values[];

/** Array size allocated           */
private int m_allocatedSize;

/** Index into the array of booleans          */
private int m_index;

/**
 * Default constructor.  Note that the default
 * block size is very small, for small lists.
 */
public BoolStack()
{
  this(32);
}

/**
 * Construct a IntVector, using the given block size.
 *
 * @param size array size to allocate
 */
public BoolStack(int size)
{

  m_allocatedSize = size;
  m_values = new boolean[size];
  m_index = -1;
}

/**
 * Get the length of the list.
 *
 * @return Current length of the list
 */
```

```
public final int size()
{
  return m_index + 1;
}

/**
 * Clears the stack.
 *
 */
public final void clear()
{
  m_index = -1;
}

/**
 * Pushes an item onto the top of this stack.
 *
 *
 * @param val the boolean to be pushed onto this stack.
 * @return  the <code>item</code> argument.
 */
public final boolean push(boolean val)
{

  if (m_index == m_allocatedSize - 1)
    grow();

  return (m_values[++m_index] = val);
}

/**
 * Removes the object at the top of this stack and returns that
 * object as the value of this function.
 *
 * @return     The object at the top of this stack.
 * @throws  EmptyStackException  if this stack is empty.
```

```
 */
public final boolean pop()
{
  return m_values[m_index--];
}

/**
 * Removes the object at the top of this stack and returns the
 * next object at the top as the value of this function.
 *
 *
 * @return Next object to the top or false if none there
 */
public final boolean popAndTop()
{

  m_index--;

  return (m_index >= 0) ? m_values[m_index] : false;
}

/**
 * Set the item at the top of this stack
 *
 *
 * @param b Object to set at the top of this stack
 */
public final void setTop(boolean b)
{
  m_values[m_index] = b;
}

/**
 * Looks at the object at the top of this stack without removing it
 * from the stack.
 *
```

```
     * @return     the object at the top of this stack.
     * @throws  EmptyStackException  if this stack is empty.
     */
    public final boolean peek()
    {
      return m_values[m_index];
    }

    /**
     * Looks at the object at the top of this stack without removing it
     * from the stack.  If the stack is empty, it returns false.
     *
     * @return     the object at the top of this stack.
     */
    public final boolean peekOrFalse()
    {
      return (m_index > -1) ? m_values[m_index] : false;
    }

    /**
     * Looks at the object at the top of this stack without removing it
     * from the stack.  If the stack is empty, it returns true.
     *
     * @return     the object at the top of this stack.
     */
    public final boolean peekOrTrue()
    {
      return (m_index > -1) ? m_values[m_index] : true;
    }

    /**
     * Tests if this stack is empty.
     *
     * @return  <code>true</code> if this stack is empty;
     *          <code>false</code> otherwise.
     */
```

```
  public boolean isEmpty()
  {
    return (m_index == -1);
  }

  /**
   * Grows the size of the stack
   *
   */
  private void grow()
  {

    m_allocatedSize *= 2;

    boolean newVector[] = new boolean[m_allocatedSize];

    System.arraycopy(m_values, 0, newVector, 0, m_index + 1);

    m_values = newVector;
  }

  public Object clone()
    throws CloneNotSupportedException
  {
    return super.clone();
  }

}
```