**1.**

import java.io.\*;  
class ArrayStack  
{  
int top;  
int size;  
int max;  
int array[]=new int array[];//THIS IS MESSED UP  
    
ArrayStack(int t,int max)  
{  
top=t;  
size=t;  
array = new int[max];  
}  
    
boolean isEmpty()  
{  
if(top==-1)  
return true;  
else  
return false;  
}  
    
int size()  
{  
return size;  
}  
    
int peak()  
{  
return max;  
}  
    
public String toString() //CAN I HELP YOU  
{  
String a="";  
for(i=top;i<=size;i++)  
{  
int val=this.arr[i];  
a=a+val;  
}  
return a;     
}  
    
public static void main(String args[])throws IOException  
{  
ArrayStack b=new ArrayStack(1,10);  
boolean empty = b.isEmpty();  
if(empty==true)  
System.out.println("THIS STACK IS EMPTY");  
else  
System.out.println("THE STACK IS NOT EMPTY");  
    
int current=b.size();  
System.out.println("CURRENT SIZE: " + current);  
    
int max=b.peak();  
System.out.println("PEAK SIZE: " + max);  
    
String s=ob.toString();  
System.out.println("FINAL STRING: " + a);  
    
}  
}

**2.**

import java.util.Scanner;

import java.util.Stack;

public class ReverseStack

{

public static void main(String[] args)

{

String sentence;

System.out.println("Enter a sentence: ");

Scanner scan = new Scanner(System.in);

sentence = scan.nextLine();

String k = PrintStack(sentence);

}

private static String PrintStack(String sentence)

{

String reverse;

String stringReversed = "";

Stack<String> stack= new Stack<String>();

sentence.split(" ");

for(int i=0;i<sentence.length(); i++)

{

stack.push(sentence.substring(i, i+1));

}

while(!stack.isEmpty())

{

stringReversed += stack.pop();

}

System.out.println("Reverse is: " + stringReversed);

return reverse;

}

}

**3.**

import java.util.Stack;

public class Test

{

// Method to evaluate value of a postfix expression

static int evaluatePostfix(String exp)

{

//create a stack

Stack<Integer> stack=new Stack<>();

// Scan all characters one by one

for(int i=0;i<exp.length();i++)

{

char c=exp.charAt(i);

// If the scanned character is an operand (number here),

// push it to the stack.

if(Character.isDigit(c))

stack.push(c - '0');

// If the scanned character is an operator, pop two

// elements from stack apply the operator

else

{

int val1 = stack.pop();

int val2 = stack.pop();

switch(c)

{

case '+':

stack.push(val2+val1);

break;

case '-':

stack.push(val2- val1);

break;

case '/':

stack.push(val2/val1);

break;

case '\*':

stack.push(val2\*val1);

break;

}

}

}

return stack.pop();

}

**4.**

ArrayStack.java

// Represents an array implementation of a stack.

package jss2;

import jss2.exceptions.\*;

import java.util.Iterator;

public class ArrayStack<T> implements StackADT<T>

{

private final int DEFAULT\_CAPACITY = 100;

private int top; // indicates the next open slot

private transient T[] stack;

public ArrayStack()

{

top = 0;

stack = (T[])(new Object[DEFAULT\_CAPACITY]);

}

public ArrayStack (int initialCapacity)

{

top = 0;

stack = (T[])(new Object[initialCapacity]);

}

public void push (T element)

{

if (size() == stack.length)

expandCapacity();

stack[top] = element;

top++;

}

public T pop() throws EmptyStackException

{

if (isEmpty())

throw new EmptyStackException();

top--;

T result = stack[top];

stack[top] = null;

return result;

}

public T peek() throws EmptyStackException

{

if (isEmpty())

throw new EmptyStackException();

return stack[top-1];

}

public boolean isEmpty()

{

return (top == 0);

}

public int size()

{

return top;

}

public String toString()

{

String result = "";

for (int scan=0; scan < top; scan++)

result = result + stack[scan].toString() + "\n";

return result;

}

private void expandCapacity()

{

T[] larger = (T[])(new Object[stack.length\*2]);

for (int index=0; index < stack.length; index++)

larger[index] = stack[index];

stack = larger;

}

}

**5.**

public class DropOutStack<T> implements StackADT<T>

{

private int count; //number of elements in the stack

private LinearNode<T> top;

/\*Declares the maximum number of elements in the stack\*/

private final int n = 5;//max size

private LinearNode<T> prev;

private LinearNode<T> curr;

public DropOutStack()

{

count = 0;

top = null;

}

public void push(T element)

{

LinearNode<T> temp = new LinearNode<T>(element);

if (count < n) {

temp.setNext(top);

top = temp;

count++;

}

else if(count>=n && n!=1) {

prev = top;

curr = top.getNext();

while(curr != null) {

prev = prev.getNext();

curr = curr.getNext();

}

prev.setNext(null);

count--;

push(element);

}

else //if n=1

{

top.setElement(element);

}

}

public T pop() throws EmptyCollectionException

{

if (isEmpty())

throw new EmptyCollectionException("stack");

T result = top.getElement();

top = top.getNext();

count--;

return result;

}

public T peek() throws EmptyCollectionException

{

if (isEmpty())

throw new EmptyCollectionException("stack");

T result = top.getElement();

return result;

}

public boolean isEmpty()

{

return (count ==0);

}

public int size()

{

return count;

}

public String toString()

{

String result = "";

LinearNode<T> current = top;

while (current != null) {

result = current.getElement() + "\n" + result;

current = current.getNext();

}

return result;

}

}