public class Stack {

int ary [] ;

int tos;

int count ;

public Stack (int size) {

ary = new int[size] ;

count = 0 ;

tos = -1;

}

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

public boolean isFull() { return tos==ary.length-1; }

public void push(int el) throws OverFlow { if (isFull()) throw new OverFlow();

else ary[++tos] = el ; count++; }

public int pop () { if (!isEmpty()) return ary[tos--] ; count--; }

public int top () { if (!isEmpty()) return ary[tos] ; }

public void printStack() { for ( int i= 0 ; i <= tos ; i ++ ) SOP ( ary[i] ) ; }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class StackApplication {

public static void main ( String args [] ) {

Stack myStack = new Stack(6);

boolean x = myStack.isEmpty();

SOP(x);

myStack.push(34);

boolean y = myStack.isEmpty();

SOP(y);

myStack.push(34);

myStack.push(4);

myStack.push(20);

int p = myStack.pop();

SOP(p);

int q = myStack.top();

SOP(q);

myStack.printStack();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class Queue{

int ary [] ;

int rare;

int front;

int count ;

public Queue(int size) { ary = new int[size] ; rare = -1 ; front = 0 ; count = 0 ; }

Nh

public boolean isEmpty() { return count == 0 ; }

public boolean isFull() { return count == ary.length; }

public void enQueue( int el ) { if (isFull()) SOP(“Queu is Full”); else { ary[++rare] = el ; count++ ; } }

public int deQueue() { if (!isEmpty()) { count--; return ary[front++] ; } }

public int firstElement() { if (!isEmpty()) return ary[front] ; }

public void printQueue() { for ( int i = front ; i <= rare ; i ++ ) SOP (ary[i]); }

}

public class QueqeApplication {

public static void main ( String args [] ) {

Queue myQueue = new Queue(6) ;

boolean x = myQueue.isEmpty() ;

SOP(x);

myQueue.enQueue(34) ;

myQueue.enQueue(12) ;

myQueue.enQueue(18) ;

boolean y = myQueue.isEmpty() ;

SOP(y);

int p = myQueue.deQueue() ;

SOP (p) ;

int r = myQueue.deQueue() ;

SOP (r) ;

int q = myQueue.firstElement() ;

SOP (q) ;

myQueue.printQueue() ;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class TNode {

int element;

TNode left;

TNode right;

public TNode ( int el) { element = el ; left=null; right=null; }

}

public class Tree {

TNode root;

public Tree() { root=null; }

public void addElement( int x ) {

TNode n = new TNode ( x) ;

if ( root==null)

root = n ;

else {

TNode temp = root;

TNode parent = temp;

for ( temp ; temp != null ; )

{ parent = temp;

if ( x > temp.element ) temp = temp.right;

if(x< temp.element) temp = temp.left;

}

if ( parent.element > x ) parent.left = n ;

if ( parent.elemnt < x ) parent.right = n ;

}

}

public TNode searchElement( int x ) {

TNode temp = root ;

for ( temp ; temp != null ; )

{ if ( temp.element == x ) break ;

else {

if ( temp.element < x ) temp = temp.right;

if( temp.elemnt > x ) temp = temp.left;

}

}

return temp;

}

}