Stack using Linked List

```
import java.util.Scanner;
    class stack
      {
          int data;
         Node next;
         Node(int data)
     {
     this.data =data;
       this.next=null;
          }}
     Node top=null;
public void push(Scanner sc)
     {
 s.o.p("Enter a data");
  int data=sc.nextInt();
 Node newNode=new Node(
 if(top==null)
   {
     top=new_node;
     }
 else
```

```
{
new-Nod
      to
   } }
public
  {
     if
  {
s.o.p
  }
 else
  {
   ~~ke~~
   t
   }

public

   N
whil
  {
 s.o
  te
{
new-Node.next=top;
        top=new-Node;
    } }
public void pop()
    {
      if(top=null)
    {
s.o.p("Stack is ~~Full~~ Empty");
    }
  else
    {
    ~~head=~~head
      top=top.next;
      }
        }
public void Display()
      {
    Node tem=top;
while( tem !=null)
    {
s.o.p(tem.data)
  tem=tem.next; } }
```

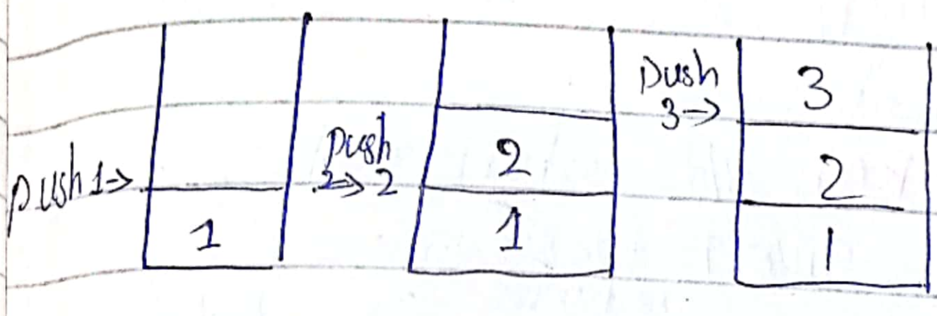
```
pu
public static void main(string arg
    {
        int d;
Scanner sc= new scanner(system.in);
  stack s= new stack();
  int l;
  do {
 s.o.p(" press 1 to push");
 s.o.p|"press 2 to pop");
 s.o.p(" press 3 to display");
 s.o.p(" Enter Your choise");
 d= sc. nextInt();
switch(d)
   {
    Case 1:
    { s.push(sc);
        break; }
   Case 2:
      { s.pop();
        break;}
    Case 3:
     {
```

```
s.disp
    bre
    }
s.o.p("En
    M
s.o.p("
  l=sc
  {wh
  s.o.p
```

```
    s.display();
        break;
         }
            }
s.o.p("Enter 0 to go back to the main
        Menu");
 s.o.p("Enter any key to Exit");
   d=sc.nextInt();
   }while(d==0);
  s.o.p("Exit sucessfully");
              } }
```

Stack

A stack can be define as a container in which insertion and deletion can be done from the one end know as the top of the stack.



standard stack operation.

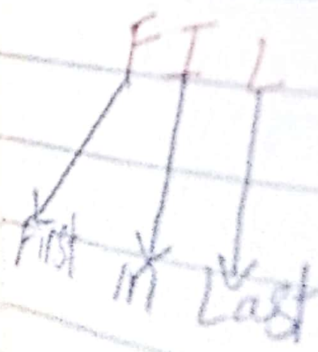
push( ): when we insert an element in a stack then the operation is know as a push. If the stack is full then the overflow condition occurs.

pop( ); when we delete an element from the stack, the operation is know as a pop. If the stack is empty means that no element exists in the stack. this state is know as underflow state.

*isEmpty(): It determine the stack is empty or not.
*isFull(); " " " Full or not.
*peek(): It returns the element at the given position.
count(): " " returns the total number of elements available in a stack.
change(): It changes the element at the given position.
display(): It prints all the elements available in the stack.

push
* Before
a stack che
* If we
element in
is full. t
condition
pop or
* Before
in a st
wealher th
* If we
element o
Empty u

Two



no
k, this
low

he

ill

element

tal number
a stack.
lement

the
e stack.

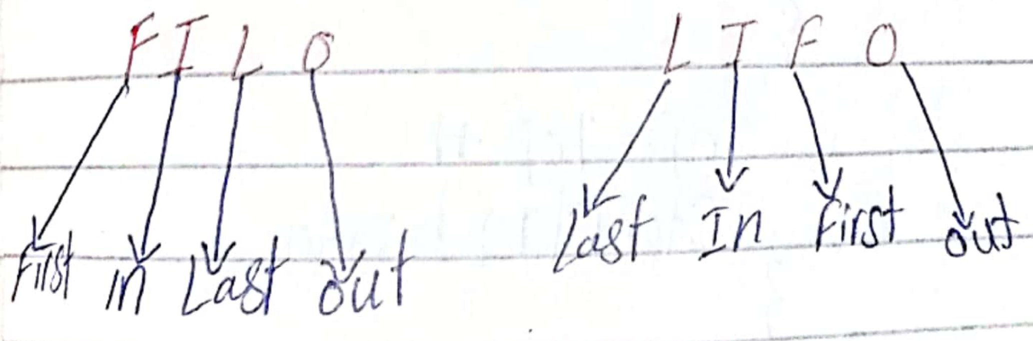
push operation

- Before inserting an element in a stack check is full.
- If we try to insert the element in a stack, and the stuck is full, then the overflow condition occure.

pop operation.

- Before deleting an element in a stack we check weather the stack is Empty.
- If we try to delete the element of the stuck when it is Empty underflow condition occure.

Two condition of Elements



Using Array Stack operation
Sodo code

```
int Max=5

int stack[Max]

int top=-1
isFull()
{
    if(top==Max-1)
        return true
    else
        return false
}
push(Data)
{
    if(!isFull())
    {
        Top=Top+1
        Stack[Top]=Data
    }
    else
        ("overflow")
}
```

```
isEmpty()
{
    if(Top==-
        return t
    else
        return
}
pop()
{
    int

        {
```

operation
e

```
isEmpty()
 {
     if(Top==-1)
      return true
      else
      return False
    }
 pop()
   {
      int    Data
             if(! isEmpty)
          {
             Data = stack[Top]
              Top=Top-1
            }
            else
             {
               "underflow" }
```