INTRODUCTION TO STACKS

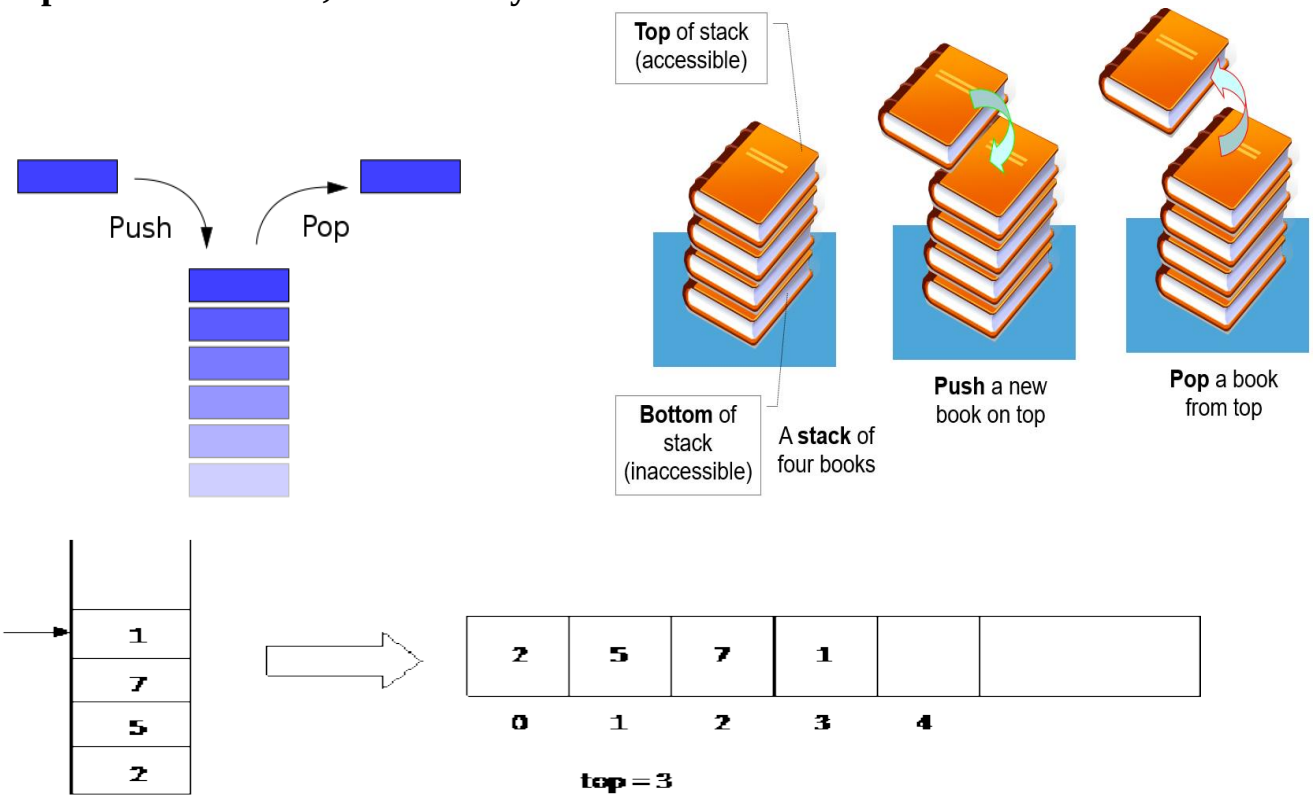
What is a stack?

Stack is a linear data structure which follows LIFO principle for insertion and deletion of data items in it.

- ➔ **LIFO means Last-In-First-Out order i.e. the last data item inserted is first one to be retrieved or deleted.**
- ➔ **A stack is always associated with only one special pointer or reference variable called 'TOP' that always refers to the topmost (*last inserted data*) data item in the stack.**

How to understand stack practically?

There are many real-life examples of a stack. Consider the simple example of plates placed over one another in a canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can easily observe that a stack follows LIFO order.



OPERATIONS ON STACKS

Mainly the following 4 basic operations are performed in the stack:

- **Push:** Adds an item at top the stack.
- **Pop:** Removes an item from top of 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:** Returns top element of stack.
- **Traversal**: visits each element of the stack only once from top to bottom.
- **Is-Empty:** Returns true if stack is empty, else false
- **Is-Full** : Returns true if the stack is full or in overflow condition.

Applications of stack:

- Conversion Infix arithmetic expressions to Postfix /Prefix expressions.
- Redo-undo features at many places like editors, photo-shops.
- Forward and backward feature in web browsers.
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver
- In Graph Algorithms like Topological Sorting and Strongly Connected Components.

Implementation of stack(static implementation)

There are two ways to implement a stack:

- Using array
- Using linked list

Array implementation of stack:

➔ In array implementation, an array of specified dimension or size is used to hold stack elements.

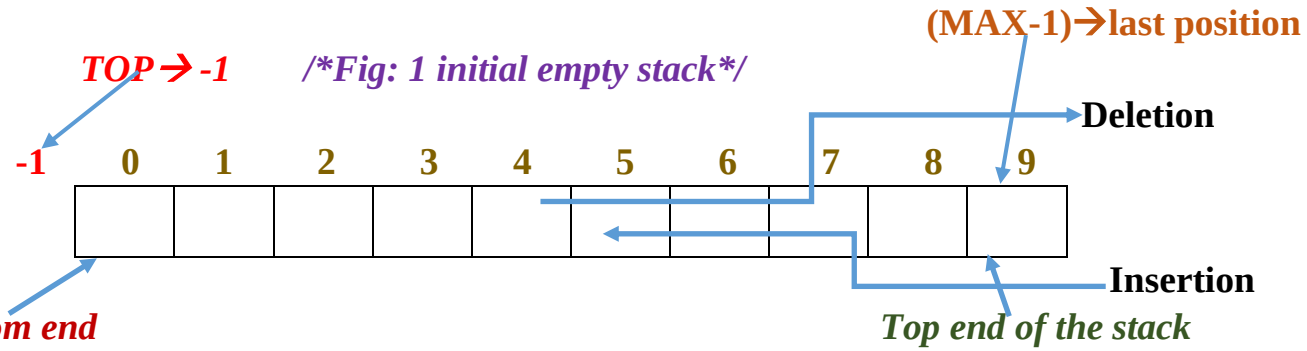
➔ Therefore before creating a stack, at first an array of specified size must be declared according to the type of data items to be stored in the array.

➔ Then an integer type special reference variable called as 'TOP' must be declared, which will always refer to the index of the topmost element present in the stack.

➔ Because the array elements are accessed by using index or position of the element in the array therefore in array implementation of stack, 'TOP' reference variable must be declared as an integer variable.

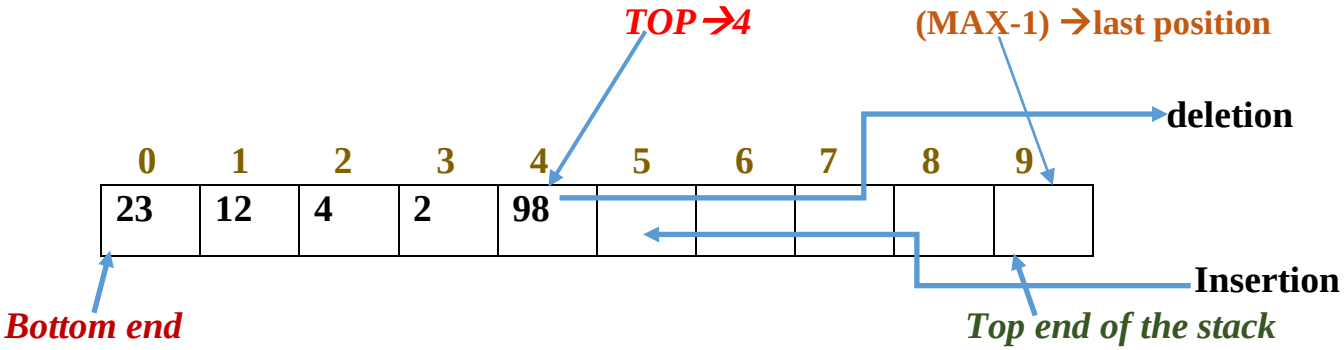
Example→1: (To store some integers in a stack)

```
static int MAX=10;  /*specifies maximum size of the stack*/
static int stack[]=new int[MAX];
stack int TOP = -1;   /*Creates an empty stack */
```



(Restricted: no insertion and deletion)

*/*Fig: 2 After insertion of some data items →Non-empty stack*/*

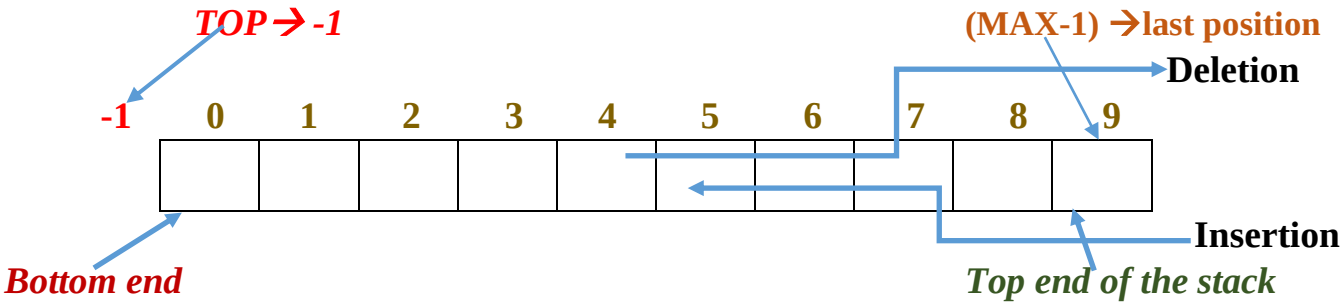


(Restricted end: no insertion and deletion)

Example→2: (Create an empty stack to store name of some persons)

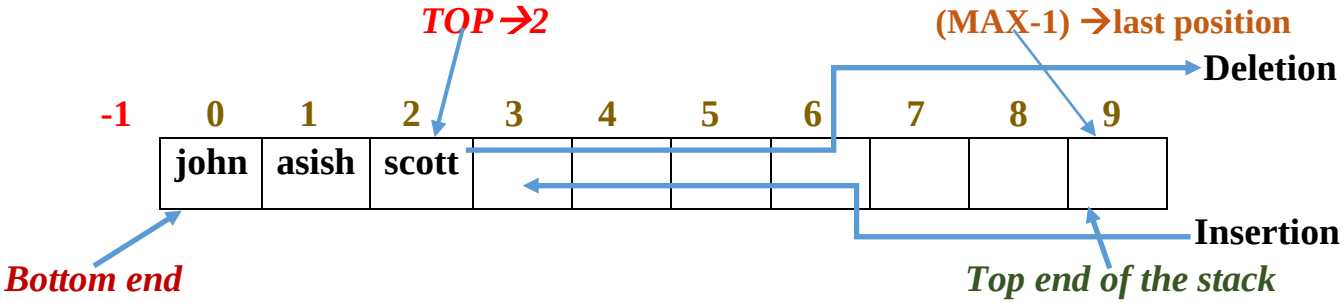
```
static int MAX=10;  /*specifies maximum size of the stack*/
static String STK[]=new String[MAX];
static int TOP = -1;   /*Creates an empty stack */
```

*/*Fig: 3 initial empty stack that will store name of some persons*/*



(Restricted: no insertion and deletion)

*/*Fig: 4 after insertion of some data items →Non-empty stack*/*

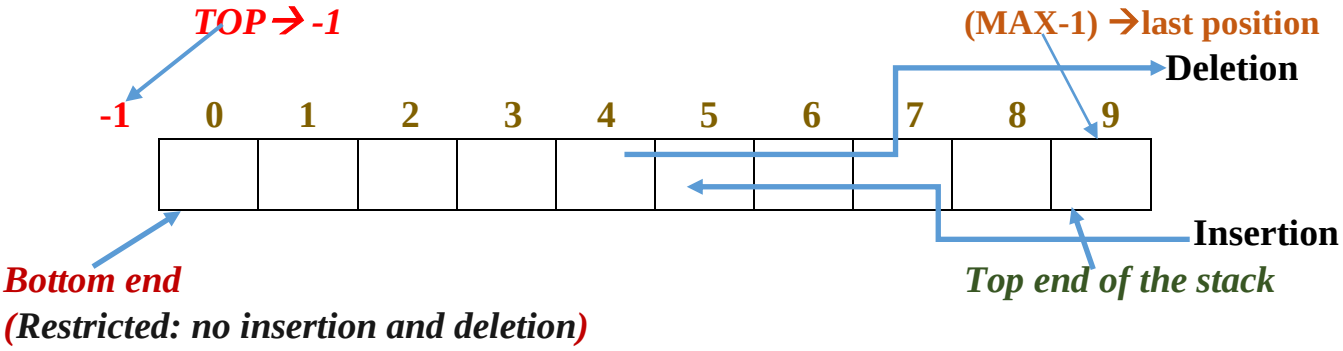


(Restricted: no insertion and deletion)

Example→3: (Create an empty stack to store single characters or grade obtained by Students)

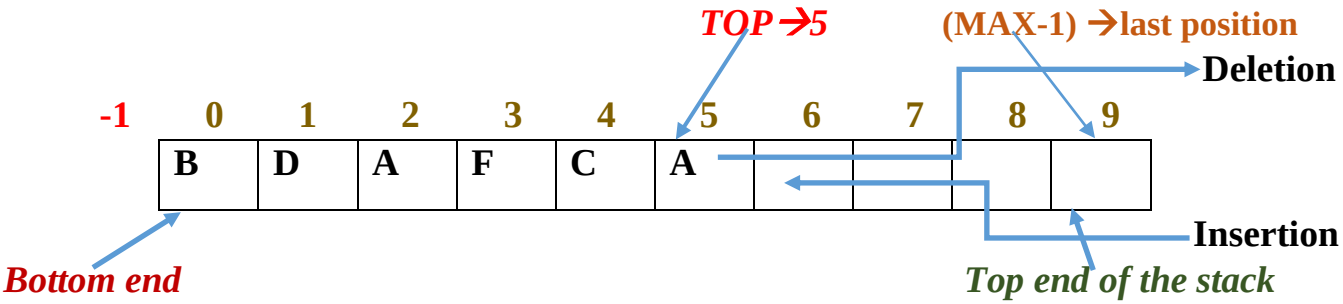
```
static int MAX=10;  /*specifies maximum size of the stack*/
static char S[]=new char[MAX];
stack int T = -1;   /*Creates an empty stack */
```

*/*Fig: 5 initial empty stack that will store grade of students*/*



(Restricted: no insertion and deletion)

*/*Fig: 6 after insertion of some data items →Non-empty stack*/*



(Restricted: no insertion and deletion)

Using the following declarations we will implement the stack using array that will hold a collection of characters or grade obtained by students:

```
static int MAX=10;  /*specifies maximum size of the stack*/
static char stack[]=new char[MAX];   /* The stack will hold single characters in each position*/
stack int TOP = -1;   /*Creates an empty stack */
```

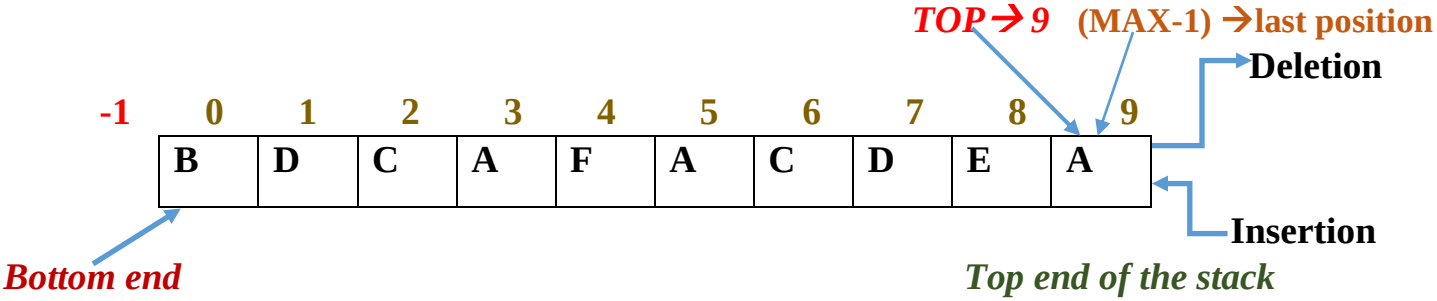
How to check overflow or is-Full condition in stack?

➔ **If the last inserted or topmost data item is present at the last index or maximum index of the array(TOP = MAX-1) that holds the stack elements then the stack will be in overflow condition i.e. if the following condition is true:**

```
If ( TOP >= MAX-1 )
    Print( " stack is full, you can't push more items");
```

/*isFull() method to check overflow condition in java*/

*/*Fig: 7 initial empty stack that will store grade of students*/*



(Restricted: no insertion and deletion)

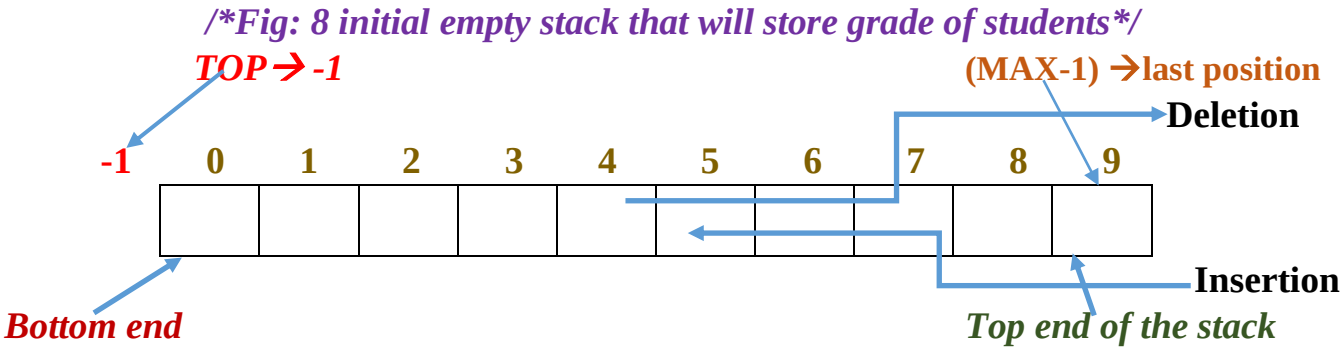
```
public static boolean isFull()
{
        if ( TOP >= MAX - 1 )
            return true;
        else
            return false;
}
/*End of isFull method*/
```

How to check underflow or is-Empty condition in stack?

➔ **If the 'TOP' pointers value is -1 i.e. it is referring to -1 position in the array that holds the stack elements, then the stack is empty i.e. if the following condition is true:**

```
If (TOP < 0 )      /*or  if ( TOP == -1) */
    Print("the stack is empty, you can't perform pop operation");
```

/*isEmpty() method to check underflow condition in java*/



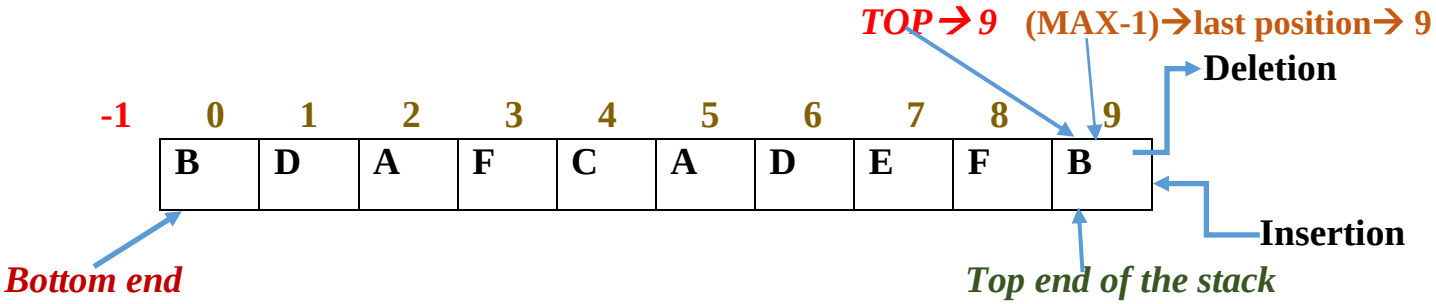

```
public static boolean isEmpty()
{
        if ( TOP <= -1 )
            return true;
        else
            return false;
}
/*End of isEmpty method*/
```

How to perform push operation?

Step-1→Check whether the stack is in overflow condition?

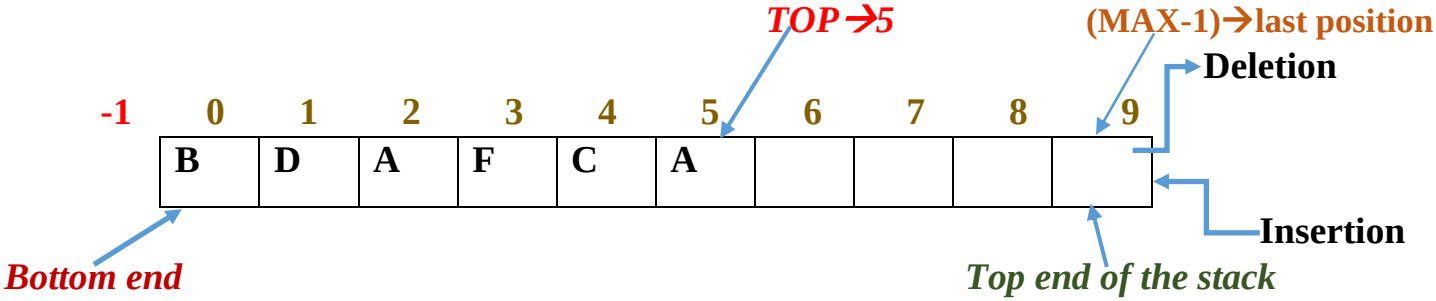
If yes: then you can't push new data items

*/*Fig: 9 the stack is in overflow condition → stack is full*/*



(Restricted: no insertion and deletion)

*/*Fig: 10 the stack is not in overflow condition→empty space available for new data items*/*

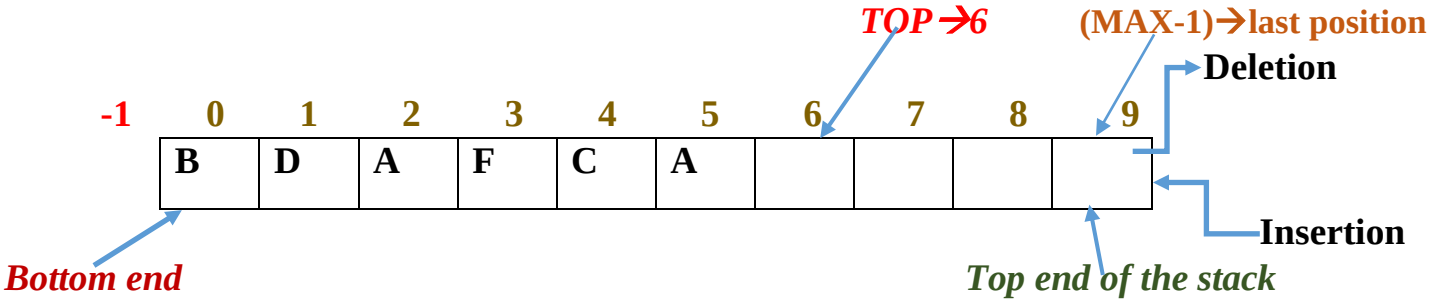


(Restricted: no insertion and deletion)

Step-2→ If no: then

a) Update the 'TOP' pointer value to next empty position i.e. TOP=TOP+1.

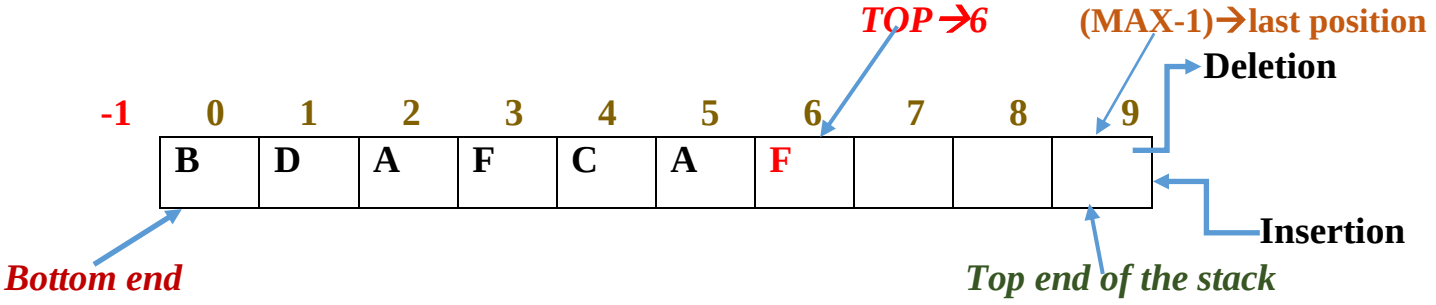
*/*Fig: 11 After updating 'TOP' pointer*/*



(Restricted: no insertion and deletion)

b) Store the new data item at the empty position referred by 'TOP' i.e.
Stack[TOP] = new_data; /* new_data→F */

*/*Fig: 12 After updating 'TOP' pointer*/*



(Restricted: no insertion and deletion)

/*PUSH() method to insert a new data item at top of the stack*/

```
public static void PUSH(char new_item)
{
        if ( isFull() == true )
        {
                System.out.println("The stack is full");
                System.out.println("You can't insert more items");
                return;
        }
        else
        {       TOP = TOP + 1;
                stack[TOP] = new_item;
        }
}
```

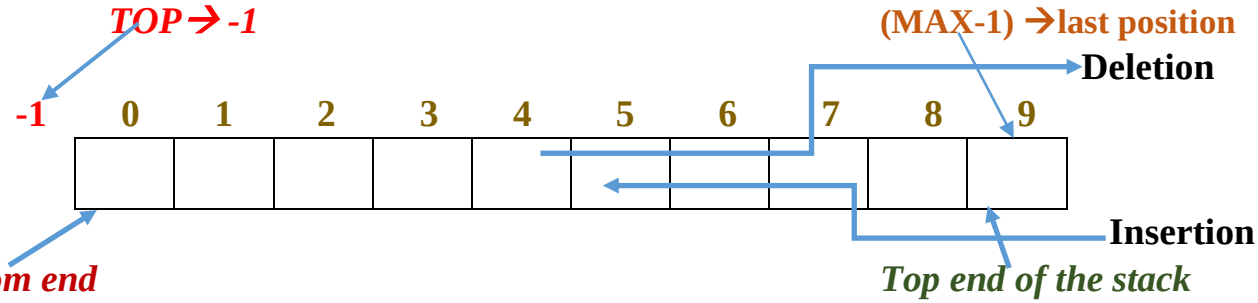
/*End of PUSH() method */

How to perform *POP* operation?

Step-1→Check whether the stack is in underflow condition i.e. empty?

If yes: you can't perform pop operation i.e. you can't delete a data item.

/*Fig: 13 initial empty stack that will store grade of students*/



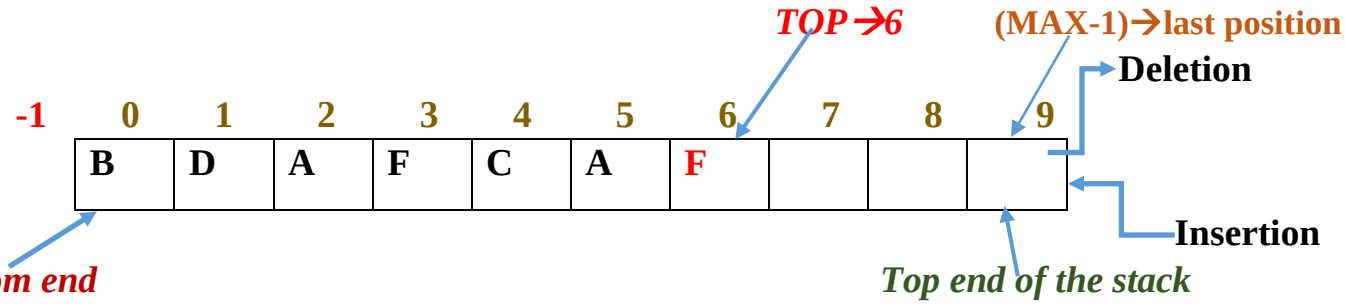
(Restricted: no insertion and deletion)

Step-2→if no: then

a) **store the element at index, to which 'TOP' pointer referring in a temporary variable**

i.e. temp = stack[TOP] /* temp→F */

/*Fig: 14 Before deletion or pop*/

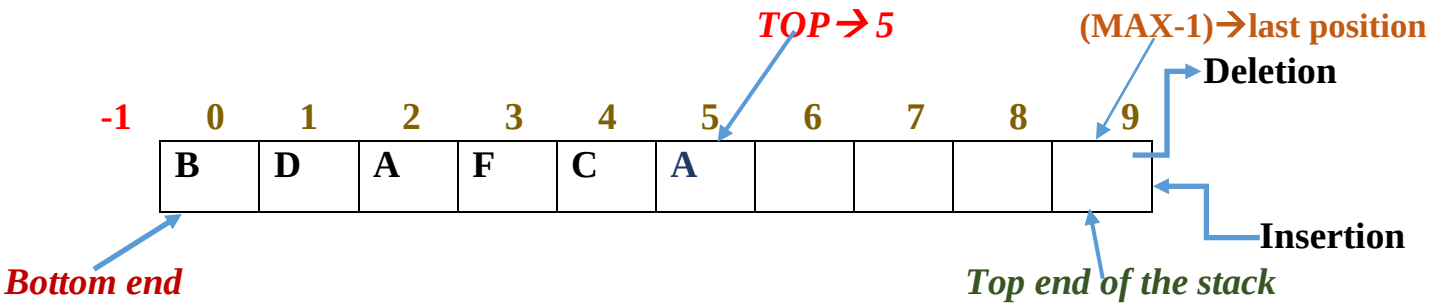


(Restricted: no insertion and deletion)

b) Update the 'TOP' pointer to previous index of the deleted elements index.

i.e. TOP = TOP – 1 /* new value of TOP→ 5 */

/*Fig: 15 After deletion or pop*/



(Restricted: no insertion and deletion)

c) print the value of temp

i.e. print("the popped item is " + temp); /*← prints F */

/*POP() method to delete the topmost data item from the stack*/

```
public static void POP()
{
        if ( isEmpty() == true )
        {
                System.out.println("The stack is empty…you can't delete an item");
                return;
        }
        else
        {       char temp = stack[TOP];
                TOP = TOP - 1;
                System.out.print("\n The popped item is : " + temp );
        }
}
```

/*End of POP() method */

How to perform *PEEK* operation?

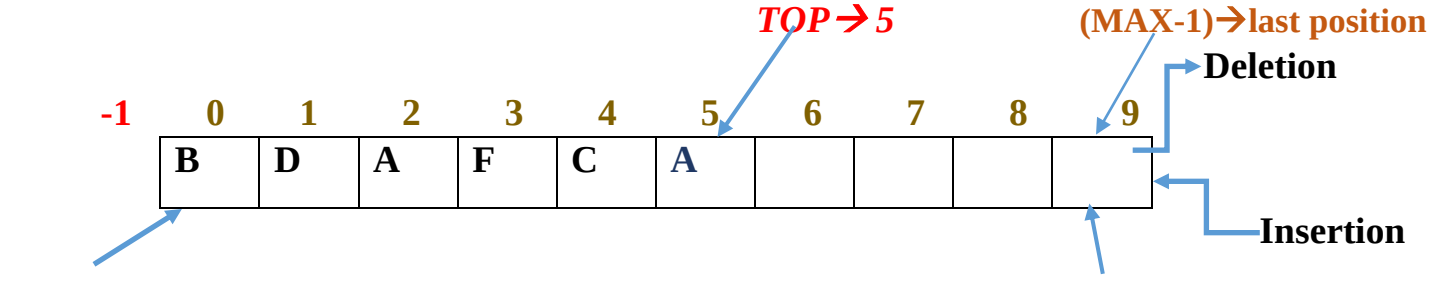
Step-1→ Check whether the stack is empty?

If yes: you can't perform peek the topmost element, stack is empty.

Step-2→ print the topmost element referred by 'TOP' pointer.

i.e. print stack[TOP] /* return stack[TOP]→ A */

/*Fig: 8 After deletion or pop*/



Bottom end ***Top end of the stack***

(Restricted: no insertion and deletion)

/*PEEK() method to display or return the topmost data item in the stack*/

```
public static void PEEK()
{
        if ( isEmpty() == true )
        {
                System.out.println("The stack is empty…you can't peek");
                return;
        }
        else
        {       char temp = stack[TOP];
                System.out.print("\n The peeked item is : " + temp );
        }
 }
```

/*End of PEEK() method */

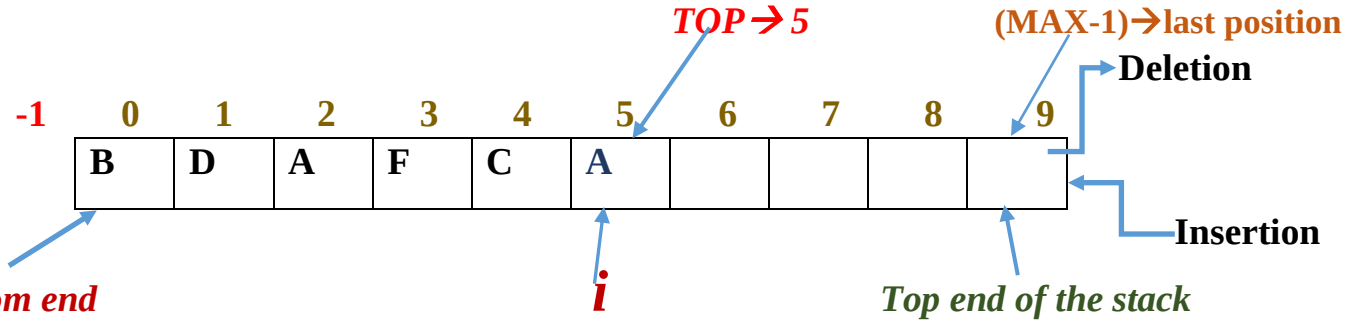
How to perform *TRAVERSE* operation?

- ➔ **This operation is same as the way you display array elements starting from the last element up-to the 0th element.**
- ➔ **So, start from the topmost most element i.e. from 'TOP' index of the stack and print the elements up-to the bottommost element at index 0.**

i.e. `for(int i= TOP ; i >= 0 ; i-- )`
`print ( stack[i] )`

Output: A C F A D B

/*Fig: 8 After deletion or pop*/



(Restricted: no insertion and deletion)

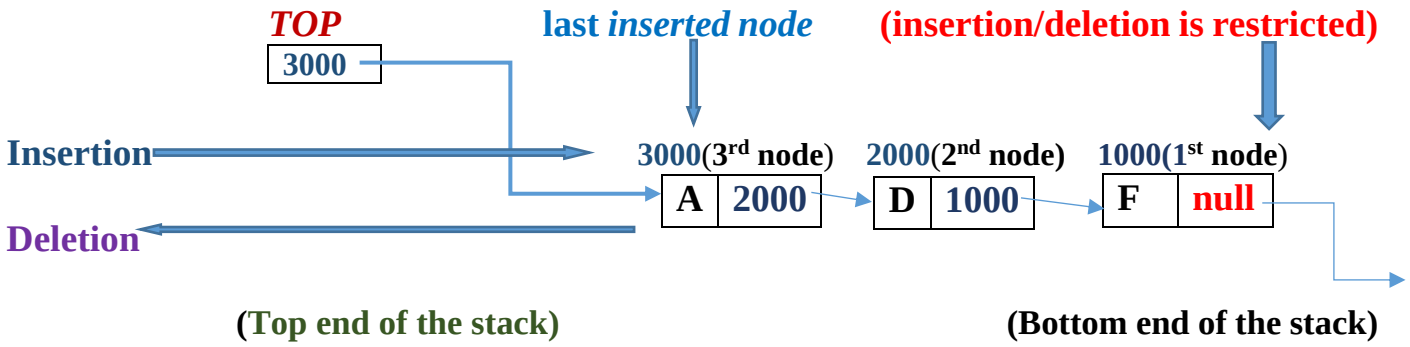
/*TRAVERSE_STACK() method to display the stack elements from top to bottom*/

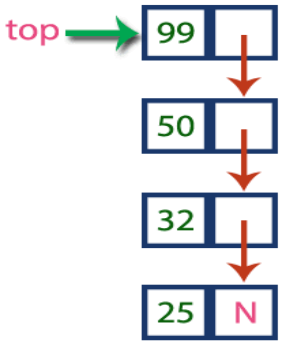
```
public static void TRAVERSE_STACK()
{
        if ( isEmpty() == true )
```

```
        {
                System.out.println("The stack is empty…you can't traverse");
                return;
        }
        else
        {   System.out.println("The stack elements from top to bottom are");
            int i = TOP ;
            while ( i >= 0 )
            {
                System.out.print(stack[i] + " → " );
                i = i - 1;
             }
        }
    }
/*End of TRAVERSE_STACK() method */
```

LINKED LIST IMPLEMENTATION OF STACK (Dynamic Implementation)

- ➔ **In linked implementation, a stack contains a linear list of data items called nodes just like a single linked list.**
- ➔ **If you restrict one end of a single linked list and perform insertion and deletion operations at the other end then it becomes a stack.**
- ➔ **The back end of the single linked, which is used to hold stack elements is considered as the bottom end of the stack, where you can't perform any operation.**
- ➔ **The front end of the list is considered as top end of the stack, where you can perform insertion or deletion operations.**
- ➔ **Here the nodes are class type variables and the 'TOP' pointer is a node type reference variable that always refers to topmost node of the stack.**





(Structure of a stack in linked list implementation)

Advantage:

→The linked list implementation of stack can grow and shrink according to the needs at runtime.

→Therefore, we can insert new nodes and delete according to requirement, so that wastage of memory space can be avoided.

Disadvantage:

→Requires extra memory due to involvement of pointer or link field in each node.

Implementation in java:

Step-1→Just like single linked list implementation declare node type class to create nodes of the stack according to data items you want to store in each node.

Example→1: (To store only an integer value in each node of the stack)

```
Class STACK_NODE
{
    int info ;
    STACK_NODE link;
}
```

Example→2: (To store only a single character value in each node of the stack)

```
Class STACK_NODE
{
    Char info;
    STACK_NODE link;
}
```

Example→3: (To store name of a person in each node of the stack)

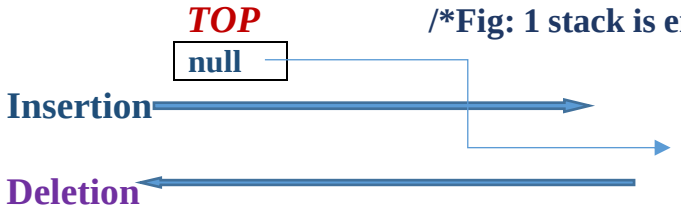
```
Class STACK_NODE
{
    String name;
    STACK_NODE link;
}
```

Step-2→: Just like single linked list implementation, declare a STACK_NODE type reference or pointer variable called 'TOP' in the main class where 'main' method resides, which will create an empty stack initially.

NOTE: We will use the following class declaration to create nodes where each node of the stack will hold a single character value or grade of students in the information part:

For example:

```
Class STACK_NODE
{
      char info ;
      STACK_NODE link;
}
public class STACK_LL_DEMO
{
       static STACK_NODE TOP = null ; /*←Creates an empty stack*/
```



/*Fig: 1 stack is empty */ (insertion/deletion is restricted)

(Top end of the stack) (Bottom end of the stack)

```
       public static void main(String[] args)
       {
              /*body of main method is to be written*/
       }
      /*Define all other methods on stack*/

} /* END OF STACK_LL_DEMO CLASS*/
```

/*isFull method to check whether the stack is in overflow condition */

- **Here we will not define any specific method to check overflow condition.**
- **Because, dynamic data structure has no maximum size like arrays where you use the condition top>=max-1 to check overflow.**
- **In dynamic implementation of any data structure, memory gets allocated during run time as you create nodes.**
- **So whenever your computer memory will vanish after allocation of memory continuously, you can't create more nodes.**
- **Therefore when you write a statement to create a node using the node class data type, 'new' operator will return 'null' value if it can't allocate memory when the memory is full, but if the new operator allocates memory for the node when memory is not full then it returns address of the node.**
- **We can use the above concept to check overflow condition in dynamic data structures while creating nodes, which is as follows:**

For example:

```
STACK_NODE newnode= new STACK_NODE(); /*← creates node*/
```

After the execution of the above statement if the following condition will be true than we can say memory is full which nothing but the overflow condition:

```
if ( newnode == null )
{
    System.out.println("STACK IS IN OVERFLOW CONDITION");
    System.out.println("new operator could not allocate memory");
    System.out.println("Because your computer memory is full");
    return;
}
```

/*isEmpty method to check whether the stack is in underflow condition empty */

```
public static boolean isEmpty()
{
    if ( TOP == null )
        return true;
    else
        return false;
}
```

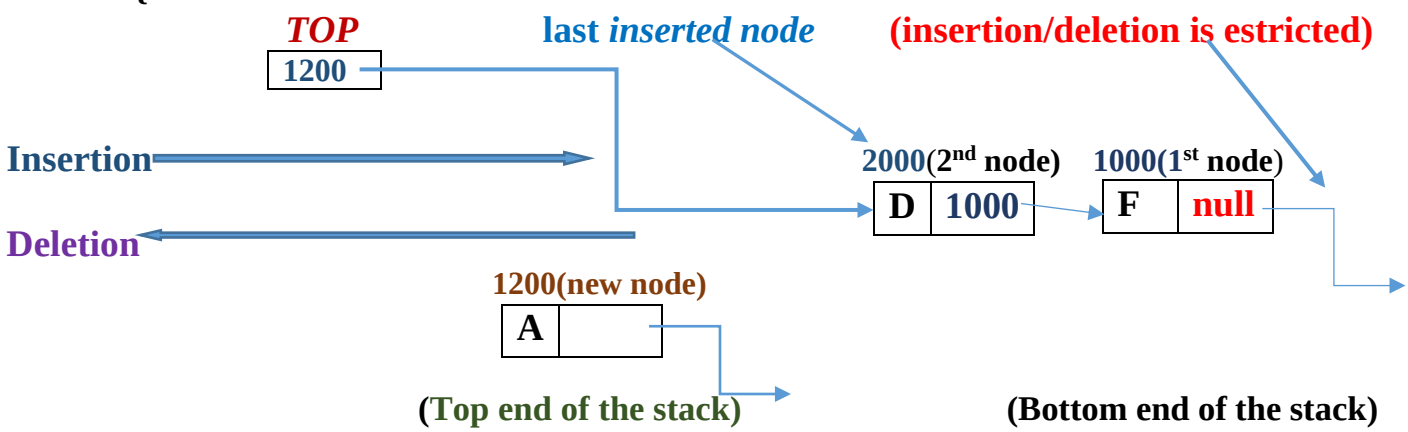
/*End of isEmpty() method*/

How to perform push operation?

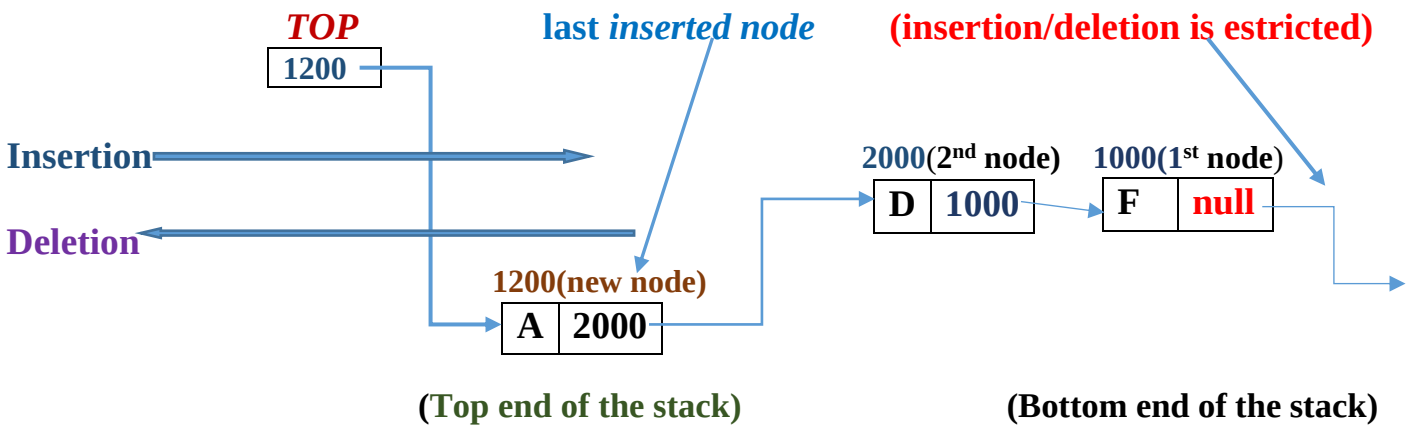
/*PUSH() METHOD*/

```
public static void PUSH( int item)
{
    STACK_NODE newnode= new STACK_NODE();

    if ( newnode == null )
    {
        System.out.println("The memory full");
        System.out.println("You can't push more items");
        return;
    }
    else    /*Fig:1 Before push operation or insertion of the new node */
    {
```



/*Fig:2 After push operation or insertion of the new node */



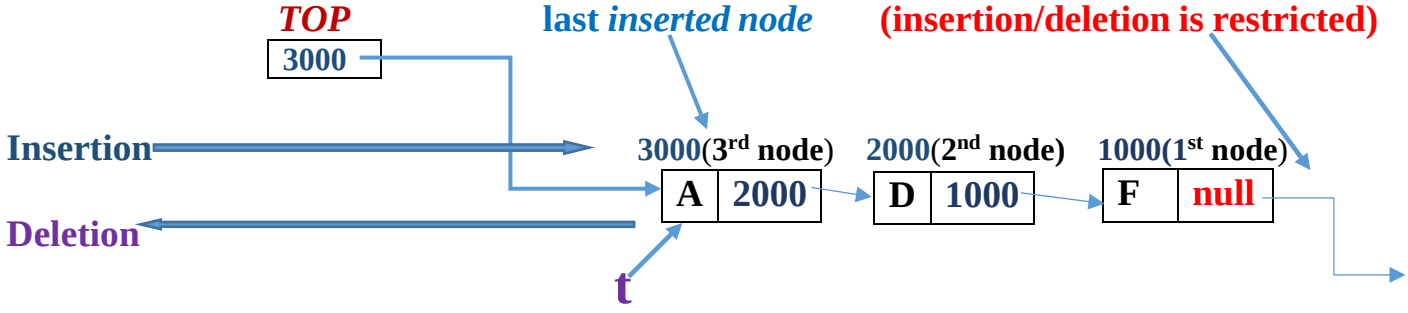

```
        }   /*End of  'else' clause*/

    } /* End of PUSH() Method*/
```

/*POP operation : to delete the topmost or the last inserted node from stack*/

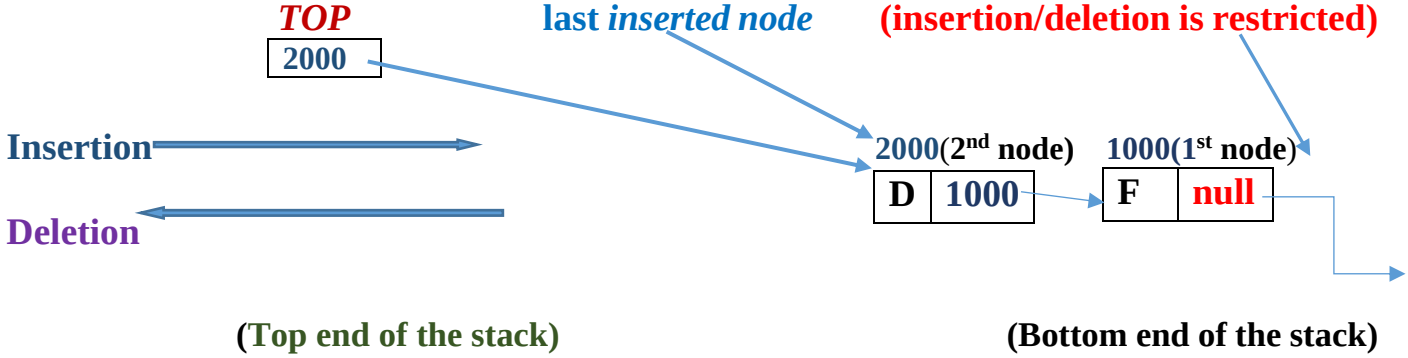
```
    public static void POP()
    {       if ( isEmpty() == true )
            {
                    System.out.println("The stack is empty…you can't pop");
                    return;
            }
            else
            {       STACK_NODE t = TOP;
```

/*Fig: 3Before deletion of the topmost node i.e. 3rd node in this figure */




```
                TOP = TOP.link ;
                t.link = null;
```

/*Fig: 4 after deletion of the topmost node i.e. 3rd node in this figure */



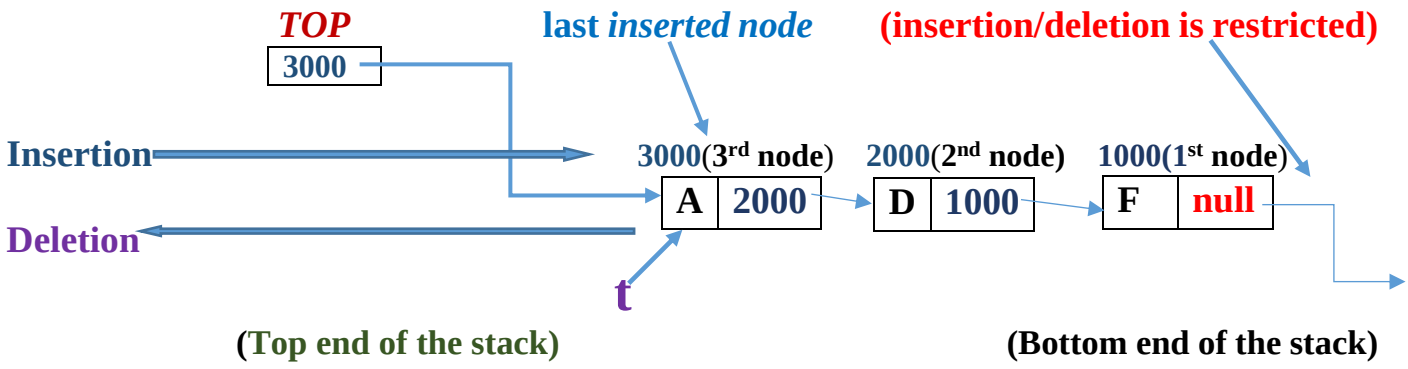
```
                System.out.print("\n The popped item is : " + t.info );
        } */ End of else clause*/

    } /*End of POP method*/

/* PEEK() method that will print or return the topmost nodes value or info*/

    public static void PEEK()
    {
        if ( isEmpty() == true )
        {
            System.out.println("The stack is empty..you can't peek");
            return ;
        }
        else
        {   char t = TOP.info ;
            System.out.print("\n The peeked item is : " + t );
        }
     } /* End of peek() method*/
```

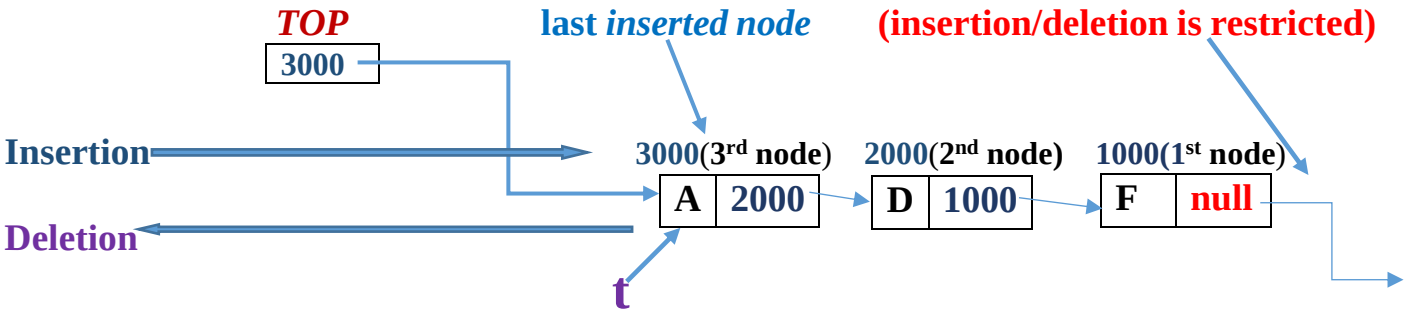
/* Fig: 5 The final stack we have created */




```
/* TRAVERSE_STACK() method to visit each node of the stack*/

public static void TRAVERSE_STACK()
    {
        if ( isEmpty() == true )
        {
            System.out.println("The stack is empty..you can't pop");
            return ;
        }
        else
        {   System.out.println("The stack elements from top to bottom are");
            STACK_NODE t = TOP ;
            while ( t != null )
            {
                System.out.print( t.info + " -> " );
```

```
            t = t.link;
        } /*End of while…loop*/
/*Fig: 6 the final stack that we have created*/
```



(Top end of the stack) (Bottom end of the stack)

```
    }/*End of else clause*/
} /*End of traverse method*/
```