Stack:-

Stack is a linear data structure in which elements are added or removed only at the end, called the top of the stack.
Hence there is no any to add or delete elements anywhere else in the stack.
A stack work is based on the last-in-first-out principle, which means the data is inserted last into the stack, and that data will remove first.

Example:-

A stack of plates.
A stack of books.

common operations on stack:-

push():-By using this method we a new record inserting into the stack.
pop():-This method remove and return the top element of the stack.
peek():-By using this method we get the topmost element of the stack.
isEmpty():-This method returns true if the stack is empty.
size():-This method returns the size of the stack.
search():-This method returns true if the given element existed in the stack.

There are two ways to implement the stack:-

1. implementation of stack using arrays.
2. implementation of stack using linked list.

Implementation of stack using arrays:-

```
package com.vikas.ds;
public class StackArray
{
int size = 5;
int[] data;
int top = -1;
StackArray(){
data = new int[size];
}
boolean isEmpty(){
return top==-1;
```

```
}
int getSize()
{
return top+1;
}
void print()
{
if(isEmpty())
{
System.out.println("stack under flow");
return;
}
Else
{
for(int i=0;i<=top;i++)
System.out.print(data[i]+" ");
System.out.println();
}
}
void push(int value){
if(getSize()==data.length){
System.out.println("stack over flow");
return;
}
else{
top++;
data[top] = value;
}
}
int pop(){
if(isEmpty()){
System.out.println("stack is under flow");
return -1;
}
else{
int value = data[top];
top--;
return value;
}
}
int peek(){
if(isEmpty()){
System.out.println("stack is under flow");
```

```
return -1;
}
else{
return data[top];
}
}
boolean search(int value){
if(isEmpty()){
System.out.println("stack is under flow ");
return false;
}
else{
for(int i=0;i<=top;i++){
if(data[i]==value)
return true;
}
return false;
}
}
public static void main(String[] args)
{
StackArray s = new StackArray();
s.push(111);
s.push(222);
s.push(333);
s.push(444);
s.push(555);
s.print();
System.out.println(s.pop());
s.print();
System.out.println(s.peek());
System.out.println(s.search(333));
System.out.println(s.search(999));
}
}
Result:-
111 222 333 444 555
555
111 222 333 444
444
true
false
```

Implementation of stack using linked list:-

```
package com.vikas.ds;
public class StackLinkedList
{
Node head=null;
int size = 0;
class Node{
int value;
Node next;
Node(int value,Node next){
this.value = value;
this.next = next;
}
}
int getSize(){
return this.size;
}
boolean isEmpty(){
return size==0;
}
void print(){
Node temp=head;
if(isEmpty()){
System.out.println("stack is empty");
return;
}
while(temp!=null){
System.out.print(temp.value+" ");
temp = temp.next;
}
System.out.println();
}
void push(int value){
head = new Node(value,head);
size++;
}
int peek(){
if(isEmpty())
return -1;
else
return head.value;
}
int pop(){
```

```
if(isEmpty()){
System.out.println("stack is under flow");
return -1;
}
else{
int temp = head.value;
head = head.next;
return temp;
}
}
public static void main(String[] args)
{
StackLinkedList sll = new StackLinkedList();
sll.push(111);
sll.push(222);
sll.push(333);
sll.push(444);
sll.print();
System.out.println(sll.peek());//
System.out.println(sll.pop());//444
System.out.println(sll.pop());//333
sll.print();
}
}
```

Result:-

```
444 333 222 111
444
444
333
222 111
```

Application of Stack in Data Structure:-

1. Evaluation of Arithmetic Operator.
2. Backtracking.
3. Delimiter Checking.
4. Reverse the Data.
5. Processing Method Call.

Evaluation of Arithmetic Operator:- A arithmetic expression contains operators, operands, and parenthesis(Left parenthesis and right parenthesis).

So we evaluate an arithmetic expression we required follows given two steps:-

Step:-1 First convert the given expression into special notation.

Step:-2 Evaluate the expression in the new notation.

Notation for Arithmetic Expression:- There are three notations to represent an arithmetic expression.

1. Infix Notation.
2. Prefix Notation.
3. Postfix Notation.

Infix Notation:-In this notation we write an arithmetic expression, in which each operator is placed between the operand. Infix expression contains parenthesis or not, it depends on the requirement.

Example:-
Infix Expression:-A+B

Prefix Notation(Polish-Notation):-In this notation we write an arithmetic expression, in which the operator is placed before the operand.
This notation we are also known as Polish-Notation.

Example:-
Prefix Expression:- +AB

Postfix Notation(Reverse-Polish-Notation):- In this notation we write an arithmetic expression, in which the operator is placed after the operand.
This notation we are also known as Reverse-Polish-Notation.

Why we convert from Infix-Notation to Prefix-Notation and Infix-Notation to Postfix-Notation?
There are some problems in Infix-Notation:-

1. **Remember Operator Precedence:-**Apply Operator Precedence or apply BODMAS rule.

   Operator Precedence Table:-

Operator	Symbol
Parenthesis	( ), {}, [ ]
Exponents	^
Multiplication and Division	*, /
Addition and Subtraction	+ , -

2. **Associativity:-**If two operators have the same precedence; associativity is used to determine the order of evaluation. If we go from left to right, then it is left associativity and if we go from right to left then it is right associativity.

To overcome the above problems, we used prefix or postfix notation, In both notations, we have not required to solve any parenthesis, operator precedence rule, and associativity rules.

Convert infix to postfix notation:-

Rules to convert infix to postfix notation:-

1. Scan the Infix string from left to right.
2. Initialize an empty stack.
3. If the scanned character is an operand, add it to the Postfix string.
4. If the scanned character is an operator and if the stack is empty push the character to stack.
5. If the scanned character is an Operator and the stack is not empty, compare the precedence of the character with the element on top of the stack.
6. If the top Stack has higher precedence over the scanned character, then pop the character from the stack else push the scanned character to the stack. Repeat this step until the stack is not empty and the top Stack has precedence over the scanned character.
7. Repeat 4 and 5 steps till all the characters are scanned.
8. If the incoming symbol is '(', push it on to the stack.
9. If the incoming symbol is ')', pop the stack and print the operators until the left parenthesis is found.

10. If the incoming operator has the same precedence as the top of the stack, then use the associativity rules. If the associativity is from left to right, then pop and print the top of the stack then push the incoming operator. If the associativity is from right to left, then push the incoming operator.
11. At the end of the expression, pop and print all the operators of the stack.

Convert this infix expression (B*C+D*E)+F to postfix expression.

SCAN	STACK	POSTFIX EXPRESSION
A		A
*	*	A
(	*(	A
B	*(	AB
*	*(*	AB
C	*(*	ABC
+	*(+	ABC*
D	*(+	ABC*D
*	*(+*	ABC*D
E	*(+*	ABC*DE
)	*	ABC*DE*+
+	+	ABC*DE*+*
F	+	ABC*DE*+*F
		ABC*DE*+*F+

Convert this infix expression K + L - M*N + (O^P) * W/U/V * T + Q to postfix expression.

SCAN	STACK	POSTFIX EXPRESSION
K		K
+	+	K
L	+	KL
-	-	KL+
M	-	KL+M
*	-*	KL+M
N	-*	KL+MN
+	+	KL+MN*-
(	+(	KL+MN*-
O	+(	KL+MN*-O
^	+(^	KL+MN*-O
P	+(^	KL+MN*-OP
)	+	KL+MN*-OP^
*	+*	KL+MN*-OP^
W	+*	KL+MN*-OP^W
/	+/	KL+MN*-OP^W*
V	+/	KL+MN*-OP^W*U/V
*	+*	KL+MN*-OP^W*U/V/
T	+*	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T*+
Q	+	KL+MN*-OP^W*U/V/T*+Q
		KL+MN*-OP^W*U/V/T*+Q+

Convert Infix to Prefix notation:-This conversion is exactly the same as an infix to postfix conversion we just add two extra steps at the start and at the end.

Starting step:-Reverse infix expressions like

(B*C+D*E)+F----------→F+(E*D+C*B)

Now Perform Postfix conversion on F+(E*D+C*B) infix expression like previous infix to postfix conversion.

Infix Expression	F+(E*D+C*B)
Postfix Expression	ED*CB*+F+

End steps:-Reverse Final Postfix expression like:-

Postfix Expression:-ED*CB*+F+

Perform Reverse of Postfix Expression:-+F+*BC*DE

Prefix Expression:- +F+*BC*DE

Recursion:- Recursion is the process in which a method call itself continuously.
A method call itself is called a recursive method, we can call this method any number of times based on our requirements.

In java, recursion is divided into two parts based on method call.

1. Infinite Recursion.
2. Finite Recursion.

Infinite Recursion:- The method which calls by itself an infinite number of times called infinite recursion.
In a program, if we perform infinite recursion then we will get StackOverFlowError.

Example:-

```
class Demo{
void m(){
System.out.println("Good Evening");
m();
}
public static void main(String[] args)
{
Demo d = new Demo();
d.m();
}
}
```

Result:-
Good Evening message will print multiple times then we will get error StackOverFlowError.

Finite Recursion:-A method which calls itself a finite number of times is called finite recursion.
In this recursion method execution, the method will terminate with the base case.

Base Cose:-It is a special condition, which is defined inside the recursive method.
If our base condition is true, then the recursive method call will terminate itself.

Principal of Recursion:- All recursive method must have three important laws.

1. A recursive method must call itself.
2. A recursive method must have a base case.
3. A recursive method must change its state and move toward the base case.

Advantage of Recursion:-

1) recursive algorithms are easier to write.

2) easy to solve natural big problems, like the Towers of Hanoi problem

3) reduce unnecessary function calls.

4) reduce the length of the code.

5) very useful while solving data-structure-related problems.

6) we can evaluate some expressions, infix, prefix and postfix etc.

Disadvantages of Recursion:

1) recursion uses extra stack space.

2) redundant computations

3) tracing will be difficult

4) slower in execution

5) runs out of memory (StackOverFlow Error)

Types of Recursion:- There are four types of Recursion.

1. Direct Recursion.
2. Indirect Recursion.
3. Tail Recursion.
4. Non-Tail Recursion.

Direct Recursion:- Direct-Recursion a method called direct itself again and again is called direct recursive.

Signature of direct recursive.

```
m1()
{
//some code
m1();
//some code
}
```

Indirect Recursion:- Indirect recursion supposes a method m1 call m2 method and m2 method call m1 method this type of recursion is known as indirect recursion.

Signature of indirect recursive.

```
m1()
{
//some code
m2();
//some code
}
m2()
{
//some code
m1();
//some code
}
```

Tail Recursion:- A method calling itself and this method calling statement must be the last statement of a method, this recursion is called Tail Recursion.

Example:-

```
int m1(int n)
{
if(n==1)
{
return 0;
}
else
{
System.out.println(n);
```

```
return m1(n-1);
}
}
```

Non-Tail Recursion:- A method calling itself and this method calling statement should not be the last statement, this recursion is called non-tail recursion.

Example:-

```
int m1(int n)
{
if(n>0)
{
m1(n-1);
System.out.println(n);
}
}
```

Difference between recursion and iteration.

Recursion	Iteration
1. Recursion will terminate when the base condition is true.	1. Iteration statement will terminate when the condition is false.
2.Its work on method calling concept.	2.Its work on looping statement concept.
3. Extra stack memory required .	3. No stack memory required.
4.Very less code required to perform any repetitive operation.	4.Bigger code required to perform any repetitive operation.

01. Write a program to print natural numbers from 1 to n by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static void print(int n){
if(n>=1)
{
//System.out.print(n+" "); //===> n, n-1, n-2,... 1
```

```
print(n-1);
System.out.print(n+" "); // ==> 1, 2, 3, 4, .... n
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter a Number");
int n = obj.nextInt();
Demo.print(n);
}
}
```

Result:-

```
Enter a Number
5
1 2 3 4 5
```

02. Write a program to calculate sum of 'n' natural numbers by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int sum(int n){
if(n==1)
{
return 1;
}
else
{
return n+sum(n-1);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter a Number");
int n = obj.nextInt();
System.out.println(Demo.sum(n));
}
}
```

Result:-

```
Enter a Number
10
55
```

03. Write a program to calculate a^b (a to the power b) by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int power(int a,int b)
{
if(b>=1)
{
return a*power(a,b-1);
}
else
{
return 1;
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter Cofficient");
int a = obj.nextInt();
System.out.println("Enter Exponent");
int b = obj.nextInt();
System.out.println("Result="+Demo.power(a,b));
}
}
```

Result:-

```
Enter Cofficient
5
Enter Exponent
3
Result=125
```

4. Write a program to find the factorial of the given number by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int fact(int n)
{
if(n==0)
{
return 1;
}
```

```
else
{
return n*fact(n-1);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter Number");
int n = obj.nextInt();
System.out.println("Sum="+Demo.fact(n));
}
}
```

Result:-

```
Enter Number
4
Sum=24
```

07. Write a program to check whether the given number is a prime number or not by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static boolean isprime(int n,int i)
{
if(i==1)
{
return true;
}
else if(n%i==0)
{
return false;
}
else
{
return isprime(n,--i);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter Number:");
int n = obj.nextInt();
System.out.println(Demo.isprime(n,n/2));//true or false
}
}
```

Result:-

```
Enter Number:
11
True
```

08. Write a program to find a sum of digits present in the given number by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int sumofdigits(int n)
{
if(n==0)
return 0;
else
return (n%10)+sumofdigits(n/10);
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter n value:");
int n = obj.nextInt();
System.out.println(Demo.sumofdigits(n));
}
}
```

Result:-

```
Enter Number:
8888
32
```

09. Write a program to calculate the reverse of the given number by using recursion.

```
Formula to find reverse :-((n%10)*pow(10,len-1))+rev(n/10,--len)

n=98123, len=5 ----> 3*pow(10,4) + rev(9812,4) ----> 3*10000=30000

n=9812, len=4 -----> 2*pow(10,3) + rev(981,3) -----> 2*1000 = 2000

n=981, len=3 ------> 1*pow(10,2) + rev(98,2) ------> 1*100  =  100

n=98, len=2 -------> 8*pow(10,1) + rev(9,1) -------> 8*10   =   80

n=9, len=1 --------> 9*pow(10,0) + rev(0,0) -------> 9*1    =    9

n=0 ---------------> terminate --------------------->         32189

rev(98123) = 32189
```

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int reverse(int n,int len)
{
if(n==0)
{
return 0;
}
else
{
return ((n%10)*(int)Math.pow(10,len-1)) + reverse(n/10,--len);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter Value");
String s = obj.nextLine();
System.out.println("Reverse Result:-
"+Demo.reverse(Integer.parseInt(s),s.length()));//reverse of 'n'
}
}
```

Result:-

```
Enter Value
12345
Reverse Result:-54321
```

10. Write a program to count the number of digits present in the given number by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int c=0;
static int count(int n)
{
if(n!=0)
{
c++;
count(n/10);
}
return (c!=0)?c:1;
}
public static void main(String[] args)
```

```
{
Scanner obj = new Scanner(System.in);
System.out.print("Enter Number:::");
int n = obj.nextInt();
System.out.println("Number Of Digit::"+Demo.count(n));
}
}
```

Result:-

```
Enter Number:::12346
Number Of Digit::5
```

11. Write a program to convert decimal number into binary by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int convert(int n)
{
if(n==0)
{
return 0;
}
else
{
return (n%2+10*convert(n/2));
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter Decimal Value::");
int n = obj.nextInt();
System.out.println("Binary Value is::"+Demo.convert(n));
}
}
```

Result:-

```
Enter Decimal Value::10
Binary Value is::1010
```

12. Implement a program to find the nth Fibonacci number by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int fib(int n)
{
if(n==0 || n==1)
{
return n;
}
else
{
return fib(n-1)+fib(n-2);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.print("Enter Number:: ");
int n = obj.nextInt();
for(int i=0;i<n;i++){
System.out.print(Demo.fib(i)+", ");
}
}
}
```

Result:-

```
Enter Number:: 10
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

13 Write a program to find the reverse of the given string using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static String strrev(String s)
{
if(s==null || s.length()<=1)//BC
{
return s;
}
return strrev(s.substring(1))+s.charAt(0);
}
public static void main(String[] args)
{
```

```
Scanner obj = new Scanner(System.in);
System.out.print("Enter any String:");
String s = obj.nextLine();
System.out.println("Reverse Result::"+Demo.strrev(s));
}
}
```

Result:-

```
Enter any String:abcdefghi
Reverse Result::ihgfedcba
```

13. Write a program to remove the given character from a string by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static String newS(String s,int index)
{
if(index<1)
{
return s.substring(0,index+1);//s.charAt(index)+"";
}
return newS(s,index-1)+"*"+s.charAt(index);
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.print("Enter any string:");
String s = obj.nextLine();
System.out.println(Demo.newS(s,s.length()-1));//abc ---> a*b*c
}
}
```

Result:-

```
Enter any string:
axbxcxxdefxghxx
abcdefgh
```

14) Write a program to return a new String, where all the adjacent characters are separated by a "*" by using recursion.

```
"hello" ----> "h*e*l*l*o"

"abc" ------> "a*b*c"
```

"ab" -------> "a*b"

```
package com.ds;
import java.util.Scanner;
class Demo
{
static String newS(String s,int index)
{
if(index<1)
return s.substring(0,index+1);//s.charAt(index)+"";
return newS(s,index-1)+"*"+s.charAt(index);
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter any string:");
String s = obj.nextLine();
System.out.println(Demo.newS(s,s.length()-1));//abc ---> a*b*c
}
}
```

Result:-

```
Enter any string:
afghjmot
a*f*g*h*j*m*o*t
```

15) Implement a program to return a new string where identical adjacent chars are separate by *.

Ex:

abc ----> abc

hello --> hel*lo

xxyy ---> x*xy*y

```
package com.ds;
import java.util.Scanner;
class Demo
{
static String newS(String s,int index)
{
if(index<1)
{
return s.substring(0,index+1);
}
if(s.charAt(index-1)==s.charAt(index))
{
return newS(s,index-1)+"*"+s.charAt(index);
```

```
}
else
{
return newS(s,index-1)+s.charAt(index);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.print("Enter any string::");
String s = obj.nextLine();
System.out.println(Demo.newS(s,s.length()-1));//abc ---> a*b*c
}
}

Enter any string::hello
hel*lo
```

16) Write a program to count the number of times, the given char occurred by using recursion.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static int count(String s,char ch,int index) //x
{
if(index<0)
{
return 0;
}
if(s.charAt(index)==ch)
{
return 1+count(s,ch,index-1);
}
else
{
return count(s,ch,index-1);
}
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter any string:");
String s = obj.nextLine();
System.out.println(Demo.count(s,'a',s.length()-1));
}
}
```

Result:-

```
Enter any string:
abcabcabc
3
```

17) Write to replace the given old character with a new character in the original string by using recursion.

```
'x' --------> 'y'

"codex" ----> "codey"

"xxhixx" ---> "yyhiyy"

"xbix" -----> "ybiy"
```

```
package com.ds;
import java.util.Scanner;
class Demo
{
static String replace(String s,int index)
{
//Base condition
if(index<0)
return "";
if(s.charAt(index)=='x')
return replace(s,index-1)+"y";
else
return replace(s,index-1)+s.charAt(index);
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter any string:");
String s = obj.nextLine();
System.out.println(Demo.replace(s,s.length()-1));
}
}
```

Result:-

```
Enter any string:
abcdefghxxaxtccx
abcdefghyyaytccy
```

Towers of Hanoi:- Tower of Hanoi is a game of rods and discs that requires a certain number of discs of different sizes to be transferred from one rod to another.

Rule of Towers of Hanoi game:-

1. We play this game with the help of only three rods source, destination, and helper.
2. Only one disk can be moved at a time.
3. No disk may be placed on top of a smaller disk.

18). Write a program to implements Towers of Hanoi game.

```
package com.ds;
import java.util.Scanner;
class Demo
{
static void towersOfHanoi(int n,String src,String helper,String
dest)
{
if(n==1){
System.out.println("Move The Disk "+n+" from "+src+" to "+dest);
return;
}
towersOfHanoi(n-1,src,dest,helper);
System.out.println("Move The Disk "+n+" from "+src+" to "+dest);
towersOfHanoi(n-1,helper,src,dest);
}
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);

System.out.println("Enter number of disks:");
int n=obj.nextInt();

Demo.towersOfHanoi(n,"S","H","D");
}
}
```

Result:-

```
Enter number of disks:
3
Move The Disk 1 from S to D
Move The Disk 2 from S to H
Move The Disk 1 from D to H
Move The Disk 3 from S to D
Move The Disk 1 from H to S
Move The Disk 2 from H to D
Move The Disk 1 from S to D
```