Recursion Complete Analysis.

Created By
Rats.

Table of contents .

```java
package Recursion_intro;

import java.util.Scanner;

public class DecreasingPrint {

    public static void Decrease(int n){
        if ( n ==0){
            return;
        }
        else{
            System.out.println(n);
            Decrease( n: n-1);
        }
    }


    public static void main (String args[]){
        System.out.println("Enter the decriesing order to print");
        Scanner scn = new Scanner(System.in);
        int i = scn.nextInt();
        Decrease(i);



    }
}
```

Problem 1 : Print Decreasing.

Print all values from given number to 0

Step1 Take the Input from the user.

** Step 2.

Call recursively to the value till it gets to "n"

for each call we need to reduce the value by 1

step3

Print all the values.

(Before or After) both ways it will work.

```java
package Recursion_intro;

import java.util.Scanner;

public class IncreasingPrint {

    public static void Acending(int n ) {
        if ( n == 0){
            return;
        }
        Acending( n: n -1);
        System.out.println(n);

    }


    public static  void main (String args[]){
        System.out.println("Enter the string you want to print till");
        Scanner scn = new Scanner(System.in);

        int n = scn.nextInt();
        Acending(n);
    }
}
```

Print Acending

Need to print all the values from 1 to n^{th} spot.

Step1 take the values input from the user.

Step2. Call for the recursive function named "Acending"

Step3 Print the values till it reaches the given value (Stop Condition)

there is one thing called as pre call & post call

```java
package Recursion_intro;

import java.util.Scanner;

public class IncreasingPrint {

    public static void Acending(int n ) {
        if ( n == 0){
            return;
        }
        Acending( n: n -1);
        System.out.println(n);

    }


    public static  void main (String args[]){
        System.out.println("Enter the string you want to print till");
        Scanner scn = new Scanner(System.in);

        int n = scn.nextInt();
        Acending(n);
    }
}
```

Part 3 →]
Part 1 →
Part 2 →

Dry Run :

Scanned the value from the user

→ n = 5

→ Called the recursive function (n)

→ if (5 = 0) false.

Acending (n-1) (5-1) = 4.

↳ if 4 = 0 (false)

Acending (3)

↳ false.

Acending (2)

false

Acending (1)

false

Acending 0

↳ True.

Once it is true then it will run the post line After the recursive call

so it will print 1 (Recursive call complete then (2)

↳ then 3

↳ 4

↳ 5

1, 2, 3, 4, 5 output-

```java
ckage Recursion_intro;

port java.util.Scanner;

blic class Increase_Decrease {

  public static void Increase (int n , int i){
      if (n == i){
          System.out.println(n);
      }
      else{
          System.out.println(n);
          Increase( n: n+1,i);
      }

  }

  public static void Decrease(int j , int i){
      if (j == 1) {
          System.out.println(j);
          Increase(j , i);
      }
      else{
          System.out.println(j);
          Decrease( j: j-1 , i);
      }

  }

  public static void main (String args[]){
      System.out.println("Enter the number to print increase and decrease");
      Scanner scn = new Scanner(System.in);
      int temp = scn.nextInt();
      int fix_temp = temp;
      Decrease(temp , fix_temp);
  }
```

Print Increase , Decrease.

Need to Increase the value to the given input and then Decrease the value to "1"

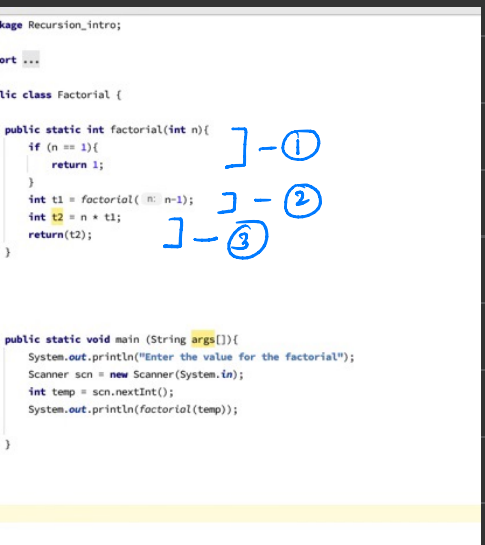
Hint : Its similar to the previous two questions if we combine the two previous question we can solve this one.

Step1 there are two input values given named temp , fix temp.

Step2 In temp we can pass (5,5) Decrease function gets called.

Step3 Decrease will run till it reaches 1

Step4 Once it reaches at 1 Increase will gets called and run till reaches 5.



```
kage Recursion_intro;

ort ...

lic class Factorial {

public static int factorial(int n){
    if (n == 1){
        return 1;
    }
    int t1 = factorial( n: n-1);
    int t2 = n * t1;
    return(t2);
}

public static void main (String args[]){
    System.out.println("Enter the value for the factorial");
    Scanner scn = new Scanner(System.in);
    int temp = scn.nextInt();
    System.out.println(factorial(temp));

}
```


Print : factorial

So, need to print the factorial of a given value from the User.

** Use the post section method of the recursive (to stop the recursion we need to check if value is equal to 1)

Post Method Runs fully After the Recursive call block :

Dry Run. Input = 5.

Check if value = 1 (false)

↳ 4 (Input) false

↳ 3 (false)

↳ 2 (false).

↳ 1 (True) Return.

Go to Step 2 → t1 = 1, n = 2
↑ Step 3 = 2 (Return)

Step 2 = t = 2, n = 3
Step 3 = 6 Return

Step 2 = t = 6 n = 4
Step 3 = 24 Return.

Step = 24 × 5 = 120 (Ans)

```java
package Recursion_intro;

import java.util.Scanner;

public class Calculatepower {

    public static int powercalculate (int val , int pow ){
        if (pow == 1){
            return val;
        }
        int temp = powercalculate(val , pow: pow -1);
        int t2 = temp * val;
        return(t2);
    }


    public static void main (String args[]){
        long start = System.currentTimeMillis();
        System.out.println("Enter the value and its power");
        Scanner scn = new Scanner(System.in);
        int val = scn.nextInt();
        int pow = scn.nextInt();

        System.out.println(powercalculate(val , pow));
        long end = System.currentTimeMillis();
        long elapsedTime = end - start;
        System.out.println(elapsedTime);


    }
}
```

-1 -2 -3

Name :→ Calculate Power.

Steps take the power & value from user

② If power is 3, value = 2 basically it means we need to self multiply the value 3 times.

(2 × 2 × 2) = 8

③ Use post Method to handle the value

Dry Run val = 2, Pow = 3

Step = 1) Check if pow is equal to 1 (false)

→ Pow - 1 = 3 - 2 = 2

check pow = 1 ? false

→ Pow - 1 = 1

check pow = 1 true

Go to Step 2 $\overset{(1)}{=}$ 1 × 2 = 2 (Return)

Go to step 2 $\overset{(2)}{=}$ 2 × 2 = 4 (Return

Go to step 3 $\overset{(3)}{=}$ 4 × 2 = 8 (Retur

(8) final Ans.

Summary: How to use the post function After the recurring call this is classic Example of that Remember to use post functionality.

```java
public static int zigzag(int n ){
    if (n == 0){
        return 1;
    }

    System.out.println( "Pre"  + n);
    zigzag( n: n -1);
    System.out.println( "In"  + n);
    zigzag( n: n -1);
    System.out.println( "Post"  + n);
    return 1;
}



public static void main (String args[]){

    System.out.println("Enter the value");
    Scanner scn = new Scanner(System.in);
    int temp = scn.nextInt();
    zigzag(temp);


}
```

Name Zigzag

Steps

Explained on the Next Page

Dry Run

When multiple values are called the Dry Run Becomes little complicated with the traditional method so we will use the method which can easily explained the overall method Eular tree (Diagram)

Summary

In this method we Understood how to Analyse multiple Recursive calls from the Eular tree Diagram.

Zig Zag Dry Run Expanded !!

```java
public static int zigzag(int n ){
    if (n == 0){
        return 1;
    }

    System.out.println( "Pre"  + n);
    zigzag( n- n -1);
    System.out.println( "In"  + n);
    zigzag( n- n -1);
    System.out.println( "Post"  + n);
    return 1;
}


public static void main (String args[]){

    System.out.println("Enter the value");
    Scanner scn = new Scanner(System.in);
    int temp = scn.nextInt();
    zigzag(temp);

}
```

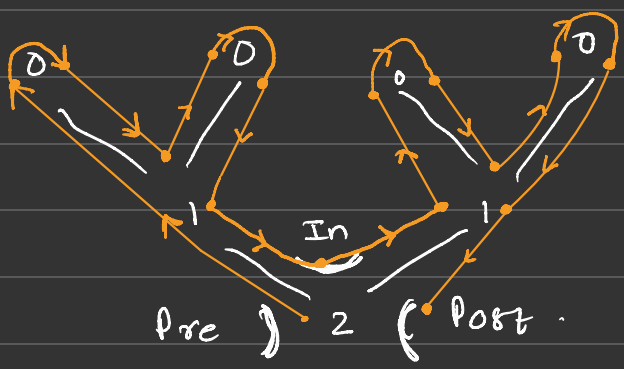
-1
-2
-3

Multiple Recursive call

To Analyse the Code better use the tree type Graph or . Gulab tree

**Very Imp Analysis

**

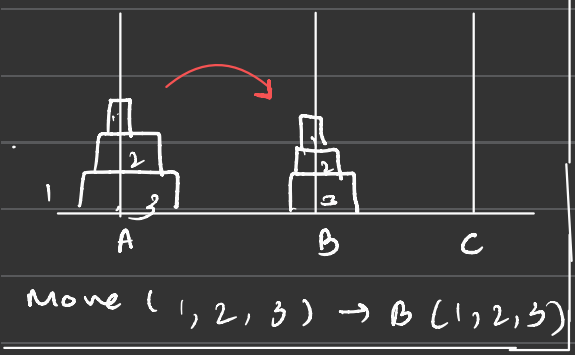


Gulabpath

By following the Gulab tree path now we can paste the values visited by the Gulab tree so the Ans will be.

(→ In 2 , Pre 1 , In 1 , Post 1 , In 2 , Pre 1
In 1 , Post 1 , Post 2 .)

with help of c move



Move (1, 2, 3) → B (1, 2, 3)

Name Tower of Hanoi

faith is important as to solve the problem

Expectation (3, A, B, C)

Dry Run

faith (2, A, B, C)

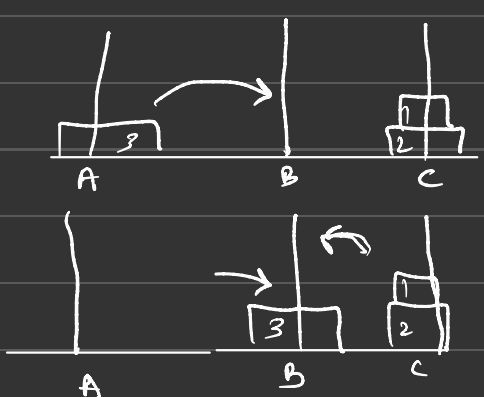
faith → Expectation

toh (3, A, B, C)

(3 Disk in A → B, help of C)

1. toh (2, A, C, B) →
2. 3 [A → B]
3. 2 [C, B, A) ←

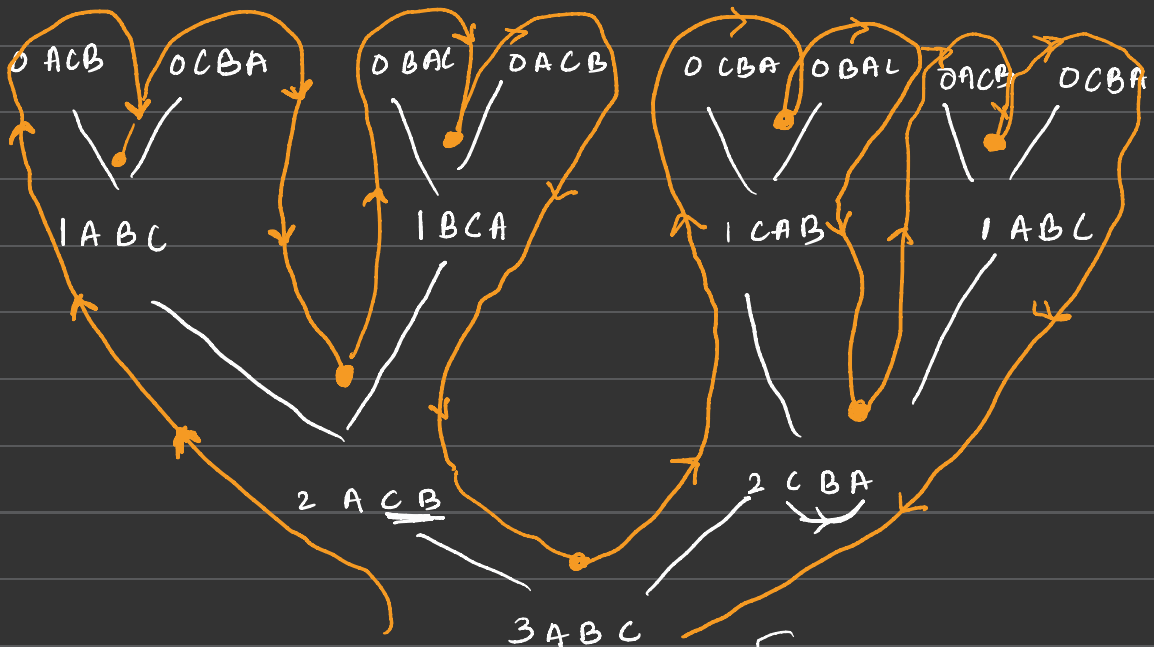
(faith)



If toh knows how to print the instructions to move 2 Disks can we make toh to print Instructions to move 3 Disks

```java
package Recursion_intro;

import ...

public class HanoiTower {

    public static void tower( int val , int t1 , int t2 , int t3 ){
        if (val == 0){
            return;
        }

        tower( val: val - 1 , t1 , t3 , t2);
        System.out.println( t1 + " -> " + t2 );
        tower( val: val -1 , t3 , t2 , t1 );

    }

    public static void main (String args[]){
        System.out.println("Enter the value ");
        Scanner scn = new Scanner(System.in);
        int value = scn.nextInt();
        int t1 = scn.nextInt();
        int t2 = scn.nextInt();
        int t3 = scn.nextInt();
        tower(value,t1,t2,t3);
```

Now if you see the code we are printing at inorder and taking calls on pre & post order.



Point (t_1 = A, t_2 = B)

1 A → B

2 A → C

1 B → C

3 A → B

1 CA

2 CB

1 AB

If we follow these instruction we can move 3 Disk from A to B using C