RECURSION

**How Function call works in Languages:**

**🡪** All the function calls in a program are stored in stack memory.

🡪 While the function is not finished executing it will remain in stack.

🡪 Main function is the first function that go into the stack and the last one to come out.

🡪 When a function finishes executing it is removed from the stack and the flow of program is restored to where the function was called.

**Recursion:**

🡪 The process of a function calling itself is called recursion.

🡪 Base Condition in recursion is the condition where recursion will stop making new calls, it is a simple if condition.

🡪 If we are calling a function again and again, you can treat it as a separate call in the stack.

🡪 Every function call takes the memory separately.

🡪 If there is no base condition function calls will keep happening and stack will be getting filled again and again and we know that every call takes memory and after certain number of calls memory of computer exceeds the limit and this gives the stack Overflow error.

**package** Recursion;

**public** **class** printNumbers {

**public** **static** **void** main(String[] args) {

//Take input n and print up to n numbers

*print1*(1);

}

**static** **void** print1(**int** n) {

System.***out***.println(n);

*print2*(2);

}

**static** **void** print2(**int** n) {

System.***out***.println(n);

*print3*(3);

}

**static** **void** print3(**int** n) {

System.***out***.println(n);

*print4*(4);

}

**static** **void** print4(**int** n) {

System.***out***.println(n);

*print5*(5);

}

**static** **void** print5(**int** n) {

System.***out***.println(n);

}

}

**package** Recursion;

**public** **class** printNumbersRecursion {

**public** **static** **void** main(String[] args) {

// print 5 numbers using recursion

*print*(1);

}

**static** **void** print(**int** n) {

**if**(n==5) {

System.***out***.println(n);

**return**;

}

System.***out***.println(n);

//This is called tail recursion

//This is the last function call

*print*(n+1);

}

}

**Why Recursion:**

🡪 It helps us in solving bigger and complex problems in simpler way.

🡪 You can convert the recursion solutions into iterations and vice versa.

🡪 Space Complexity is not constant because of recursive calls.

🡪 It helps in breaking down bigger problems into simpler problems.

**How to know that we can solve a problem in recursion:**

🡪 If we are able to break the problem into solver problems we can solve it using recursion .

🡪 For example finding the nth terms in Fibonacci series.

*Fibo(n)=Fibo(n-1)+Fibo(n-2)*

This way of writing a recursion in formula is called recurrence relation.

🡪 The base condition is represented by answers we already have. In the fibonnaci series example we know that Fibo(0)=0 and Fibo(1)=1 and this is the base condition.

**package** Recursion;

**public** **class** fibonacci {

**public** **static** **void** main(String[] args) {

System.***out***.println(*fibo*(6));

}

**static** **int** fibo(**int** n) {

// base condition

**if**(n<2) {

**return** n;

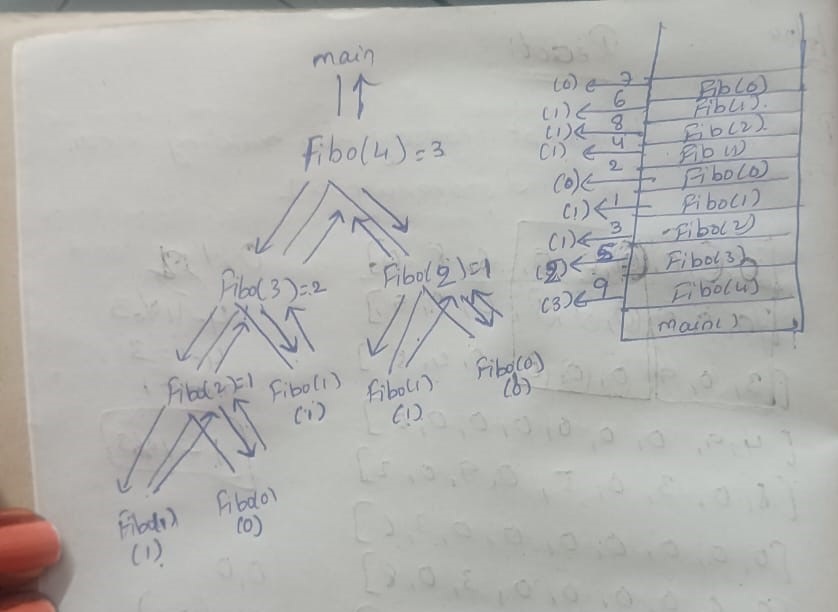
}

// This is not the tail recursion

**return** *fibo*(n-1)+*fibo*(n-2);

}

}



**How to understand and approach a problem:**

🡪 Identify if you can break down a problem into simpler problems.

🡪 Write the recurrence relation if needed.

🡪 Draw the recursive tree.

🡪 About the tree:

🡪 See the flow of function how they are getting in stack.

🡪 Identify and focus on left tree calls and right tree calls.

🡪 Draw the tree and pointer again and again using pen and paper

🡪 Use a debugger to see the flow of the program.

🡪 See how the values and what type of values are returned at each step.

🡪 See where the function call will come out of. In the end we will come out of the main function.

**Variables in Recursion:**

1. Arguments

2. Return type

3. Variables used in the body.

**Binary Search Using Recursion:**

F(n)=O(1)+F(n/2)

O(1) is for comparison we make.

**Types of Recurrence Relation:**

1. Linear Recurrence Relation🡪 Fibonacci Series.

2. Divide and Conquer Recurrence Relation🡪 Binary Search

**package** BinarySearch;

**public** **class** binarySearchUsingRecursion {

**public** **static** **void** main(String[] args) {

**int**[] arr= {2,3,4,5,6,7,8,9};

**int** target=8;

System.***out***.println(*binarySearch*(arr,target,0,arr.length-1));

}

**static** **int** binarySearch(**int**[] arr, **int** target, **int** start, **int** end) {

**if**(start>end) {

**return** -1;

}

**int** mid=start+(end-start)/2;

**if**(arr[mid]==target) {

**return** mid;

}

**else** **if**(target>arr[mid]) {

**return** *binarySearch*(arr,target,mid+1,end);

}

**else** {

**return** *binarySearch*(arr,target,start,mid+2);

}

}

}