Java Foundation with Data Structures

Topic: Recursion

Recursion

a. What is Recursion?

In previous lectures, we used iteration to solve problems. Now, we'll learn about recursion for solving problems which contain smaller sub-problems of the same kind.

Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem. By same nature it actually means that the approach that we use to solve the original problem can be used to solve smaller problems as well.

So in other words in recursion a function calls itself to solve smaller problems. Recursion is a popular approach for solving problems because recursive solutions are generally easier to think than their iterative counterparts and the code is also shorter and easier to understand.

b. How Does Recursion Work?

We can define the steps of a recursive solution as follows:

1. **Base Case:**
   A recursive function must have a terminating condition at which the function will stop calling itself. Such a condition is known as a base case.
2. **Recursive Call:**
   The recursive function will recursively invoke itself on the smaller version of problem. We need to be careful while writing this step as it is important to correctly figure out what your smaller problem is on whose solution the original problem's solution depends.
3. **Small Calculation:**
   Generally we perform a some calculation step in each recursive call. We can perform this calculation step before or after the recursive call depending upon the nature of the problem.

It is important to note here that recursion uses stack to store the recursive calls. So, to avoid memory overflow problem, we should define a recursive solution with minimum possible number of recursive calls such that the base condition is achieved before the recursion stack starts overflowing on getting completely filled.

Now, let us look at an example to calculate factorial of a number using recursion.

Example Code 1:

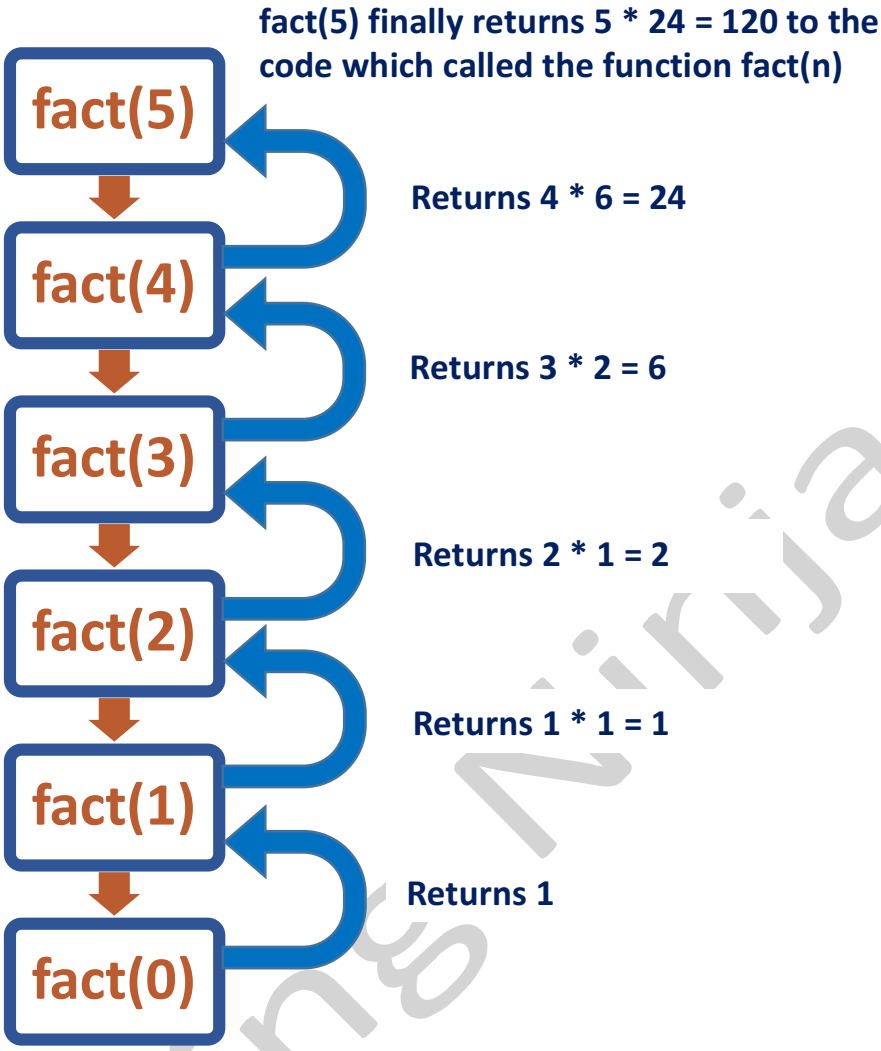
```
public class Solution{

    public static int fact(int n)
    {
      if(n==0)                          //Base Case
      {
        return 1;
      }
      return n * fact(n-1);             //Recursive call with small
    calculation
    }

    public static void main()
    {
      int num;
      Scanner s = new Scanner(Syatem.in);
      num = s.nextInt();
      System.out.println(fact(num));
      return 0;
    }
}
```

Output:

120 //For num=5

Explanation:

Here, we called factorial function recursively till number became 0. Then, the statements below the recursive call statement were executed. We can visualize the recursion tree for this function, where let n=5, as follows:



We are calculating the factorial of n=5 here. We can infer that the function recursively calls fact(n) till n becomes 0, which is the base case here. In the base case, we returned the value 1. Then, the statements after the recursive calls were executed which returned n*fact(n-1) for each call. Finally, fact(5) returned the answer 120 to main() from where we had invoked the fact() function.

Now, let us look at another example to find n^{th} Fibonacci number . In Fibonacci series to calculate nth Fibonacci number we can use the formula F(n) = F(n – 1) + F(n – 2) i.e. nth Fibonacci term is equal to sum of n-1 and n-2 Fibonacci terms. So let's use this to write recursive code for nth Fibonacci number.

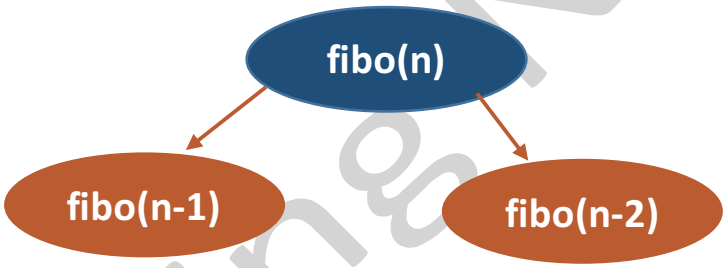
Example Code 2:

// Recursive function:

```
int fibo(int n) {
   if(n==0 || n==1) {          //Base Case
      return n;
   }
   int a = fibo(n-1);          //Recursive call
   int b = fibo(n-2);          //Recursive call
   return a+b;                 //Small Calculation and return statement
}
```

Explanation:
As we are aware of the Fibonacci Series (0, 1, 1, 2, 3, 5, 8,… and so on), let us assume that the index starts from 0, so, 5^{th} Fibonacci number will correspond to 5; 6^{th} Fibonacci number will correspond to 8; and so on.

Here, in recursive Fibonacci function, we have made two recursive calls which are depicted as follows:



Note: One thing that we should be clear about is that both recursive calls don't happen simultaneously. First fibo(n-1) is called, and only after we have its result and store it in "a" we move to next statement to calculate fibo(n – 2).

It is interesting to note here that the concept of recursion is based on the mathematical concept of **PMI** (Principle of Mathematical Induction). When we use PMI to prove a theorem, we have to show that the base case (usually for x=0 or x=1) is true and, the induction hypothesis for case x=k is true must imply that case x=k+1 is also true. We can now understand how the steps which we followed in recursion are based on the induction steps, as in recursion also, we have a base case while the assumption corresponds to the recursive call.