**What is Memoization?**

Memoization is a **concept**/**technique** for **optimization** of **functional** **programming**.

Here the idea is if we have a functional program like

Y1=g(x1),

Y2=g(x2),

Y3=g(x3) and

Y4=g(x1) again, then it’s a wasted work to recompute g(x1) if we already did it.

Hence, in sequential programming when we need to use functional programming like this we must use memorization.

**How to achieve Memoization?**

1. Use data structure.
2. Insert in data structure the fact that we have computed Y1, as the result of applying function **g to X1**.
3. Now instead of re-computing it, just look up the value from the data structure, where look up **g, X1** as input and just reuse the result.

**Note: Memoization ensure that a method should not run for same input more than once.**

**Is Memoization same as caching?**

Memoization is a specific form of caching that involves caching the return values of a function based on its parameters.

**Best use case of Memoization:**

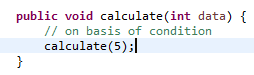
Any large computation algorithm where (**RECURSION** may use) there is a chance of computing with same parameter. Like

1: pascal’s triangle where we have binomial coefficients.

2: find out nth Fibonacci number etc.

**What is Recursion?**

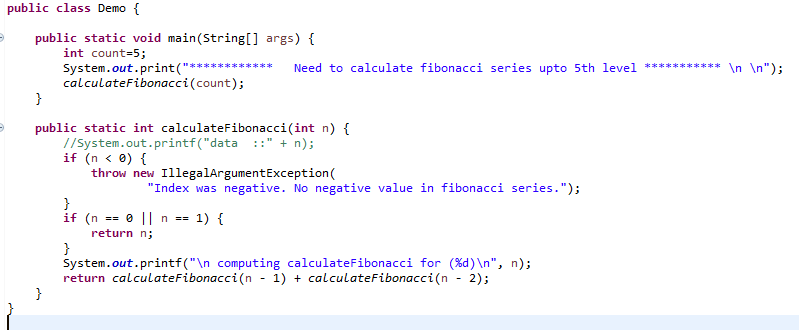
Recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem. In simpler way, **calling a function from the body itself conditionally.**

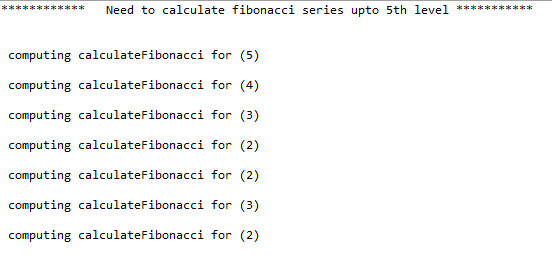


**Let’s explore Memoization with the help of Fibonacci series.**

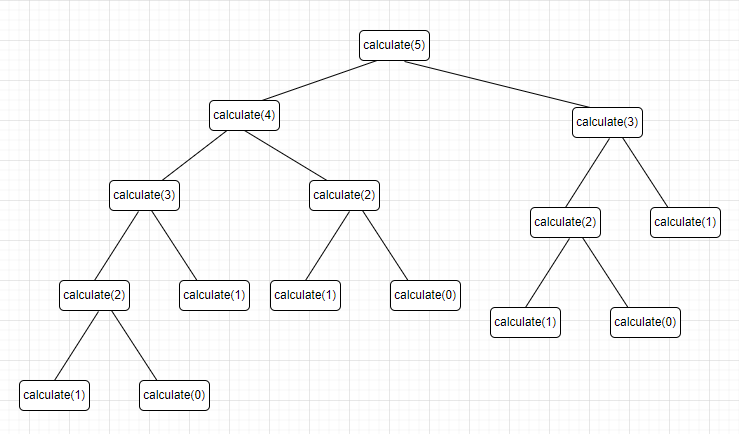
Fibonacci series: next number is the sum of previous 2 numbers like 0,1,1,2,3,5,8,13,21,34,55,89

**1: without Memoization:**





**In below image consider calculate() as calculateFibonacci()**



**Tree representation of recursive call for calculateFibonacci(5)**

**2: With Memoization:**

