

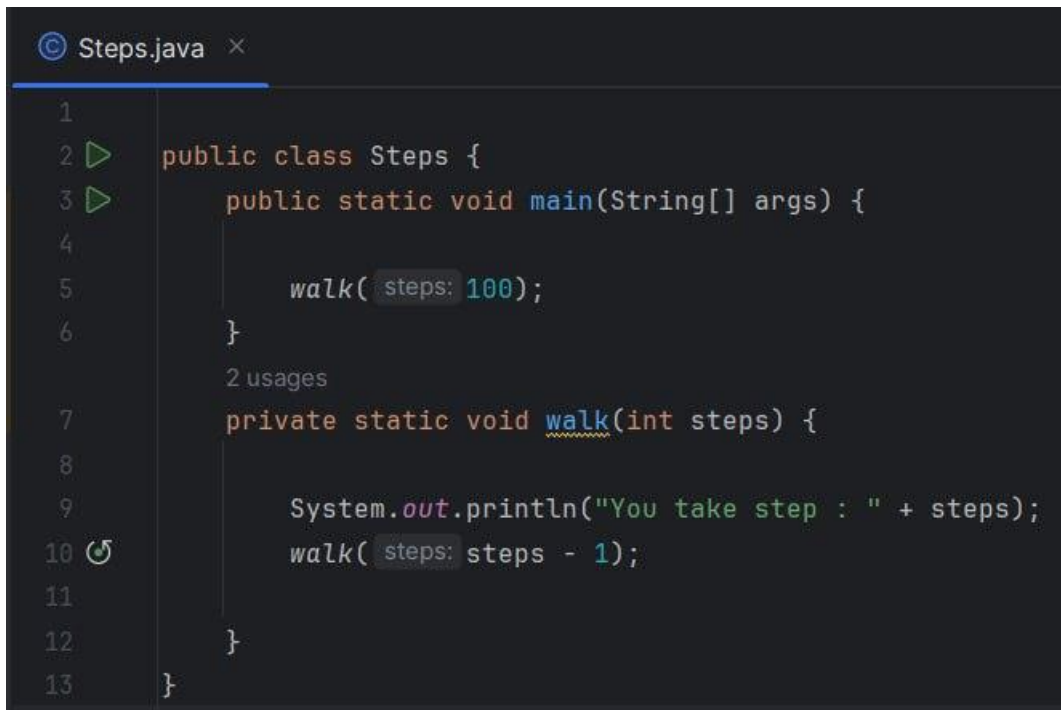
Recursion

Recursion - a function that calls itself from within helps to visualize a complex problem into basic steps.

Recursive functions and algorithms:

A common algorithm design tactic is to divide a problem into sub-problems of the same type as the original, solve those sub-problems, and combine the results. This is often referred to as the divide-and-conquer method; when combined with a lookup table that stores the results of previously solved sub-problems (to avoid solving them repeatedly and incurring extra computation time), it can be referred to as dynamic programming or memoization. To examples of lookup table we will back soon.

Ex 1 :



```
© Steps.java x
1
2 public class Steps {
3     public static void main(String[] args) {
4
5         walk( steps: 100);
6     }
7     2 usages
8     private static void walk(int steps) {
9
10        System.out.println("You take step : " + steps);
11        walk( steps: steps - 1);
12    }
13 }
```

Here we see maybe the most common example of using recursive methods, but I also specially do not write the main thing. Let's check the result :

```
You take step : 7
You take step : 6
You take step : 5
You take step : 4
You take step : 3
You take step : 2
You take step : 1
You take step : 0
You take step : -1
You take step : -2
You take step : -3
You take step : -4
You take step : -5
You take step : -6
You take step : -7
```

```
You take step : -5941
You take step : -5942
You take step : -5943
You take step : -5944
You take step : -5945
Exception in thread "main" java.lang.StackOverflowError: Create breakpoint
    at java.base/sun.nio.cs.UTF_8$Encoder.encodeArrayLoop(UTF_8.java:456)
    at java.base/sun.nio.cs.UTF_8$Encoder.encodeLoop(UTF_8.java:564)
    at java.base/java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:586)
    at java.base/sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:370)
```

The problem now named "Stack over flow Error " it is because we do not have the Base case. For our ex. It is a condition that steps can't be less than 0

```
© Steps.java x
1
2 public class Steps {
3     public static void main(String[] args) {
4
5         walk( steps: 100);
6     }
7     2 usages
8     private static void walk(int steps) {
9         if (steps == 0) {
10             return;
11         }
12         System.out.println("You take step : " + steps);
13         walk( steps: steps - 1);
14     }
15 }
```

Well now our code work correctly !

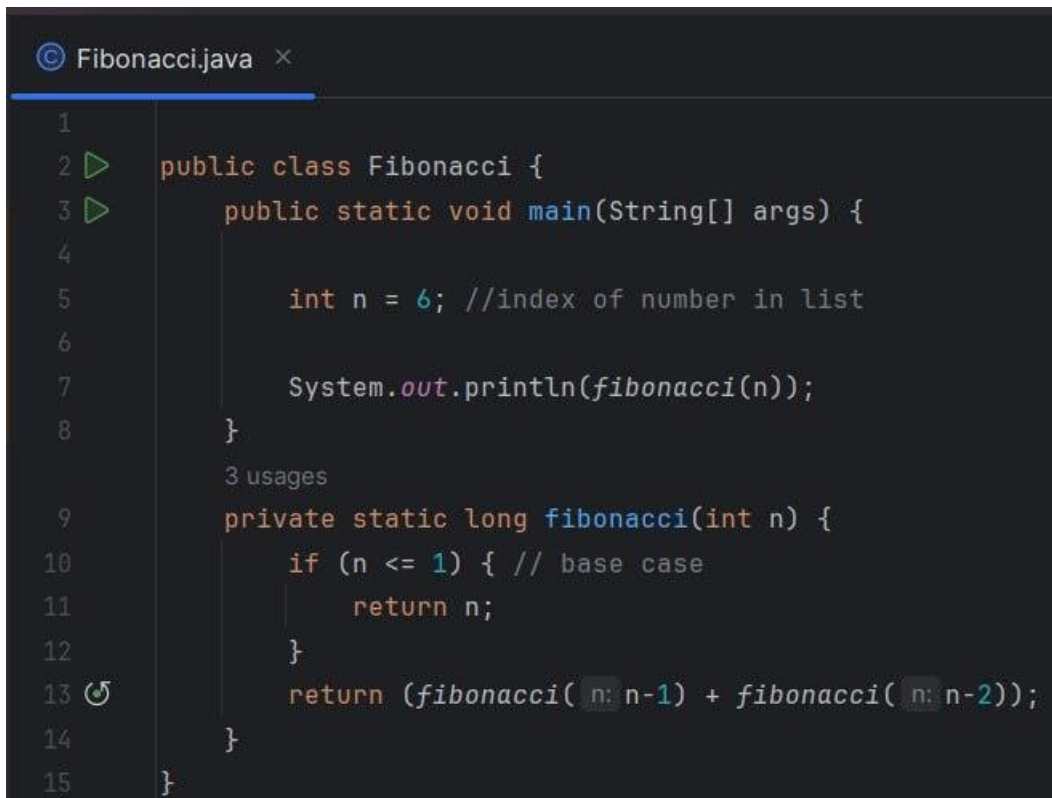
Let's do something harder :

Ex 2 (Fibonacci Series):

0 1 1 2 3 5 8 13 21 34 55 ...

$$f(n) = f(n - 1) + f(n - 2)$$

The code must find the number at its index in the list:



```
1
2 public class Fibonacci {
3     public static void main(String[] args) {
4
5         int n = 6; //index of number in list
6
7         System.out.println(fibonacci(n));
8     }
9     3 usages
10    private static long fibonacci(int n) {
11        if (n <= 1) { // base case
12            return n;
13        }
14        return (fibonacci(n-1) + fibonacci(n-2));
15    }
16 }
```

Here is the code and it looks perfect and easy to read, But what happens if I decide to find 50 index in the list – spoiler it will take approximately 2 min and it's the main disadvantage in the recursion method (much easier to read but extremely slow)

And now we should back to the lookup table. What is mean and how it use ?

Well let's check when we repeat the same work ?

As was written here our formula $f(n) = f(n - 1) + f(n - 2)$:

$$f(6) = f(5) + f(4)$$

$$f(5) = f(4) + f(3)$$

$$f(4) = f(3) + f(2)$$

As we see. We repeat the process of finding the same $f(n)$. The idea to solve this problem is same in cache intermediate result:

```
© Fibonacci.java ×
1
2 public class Fibonacci {
3
4     4 usages
5     private static long[] fibonacciCache;
6
7     6 usages
8     public static void main(String[] args) {
9
10         int n = 50; //index of number in list
11
12         fibonacciCache = new long[n+1] ;
13
14         System.out.println(fibonacci(n));
15     }
16
17     3 usages
18     private static long fibonacci(int n) {
19         if (n <= 1) { // base case
20             return n;
21         }
22         if (fibonacciCache[n] != 0 ) { // check if we already count this num before
23             return fibonacciCache[n];
24         }
25
26         long nthFibonacciNumber = (fibonacci(n-1) + fibonacci(n-2));
27         fibonacciCache[n] = nthFibonacciNumber;
28         return nthFibonacciNumber;
29     }
30 }
```

Now finding
the 50 index
take less
than 1
second !!!

To sum up:

Advantages of recursion:

1. The code may be easier to write.
2. Reduce unnecessary calling of function.
3. Extremely useful when applying the same solution.
4. Recursion reduce the length of code.
5. It is very useful in solving the data structure problem.
6. Stacks evolutions and infix, prefix, postfix evaluations etc.

Disadvantages of recursion:

1. Recursive functions are generally slower than non-recursive function.
2. It may require a lot of memory space to hold intermediate results on the system stacks.
3. Hard to analyze or understand the code.
4. It is not more efficient in terms of space and time complexity.
5. The computer may run out of memory if the recursive calls are not properly checked.

Sources :

Books:

- Data Structures and Algorithms in Java Fourth Edition
- Programming Languages: Principles and Paradigms

Websites:

- [https://en.wikipedia.org/wiki/Recursion_\(computer_science\)](https://en.wikipedia.org/wiki/Recursion_(computer_science))
- <https://www.youtube.com/watch?v=ivl5-snqul8>
- <https://www.youtube.com/watch?v=k-7jP7QFEM>
- <https://www.youtube.com/watch?v=cum3OrpURzc>
- <https://www.collegenote.net/curriculum/data-structures-and-algorithms/41/454/>