CHAPTER 13: RECURSION

13.1. What is Recursion?

Act of a method calling itself. A method that calls itself inside its method body is known as a recursive method. Kind of like the infinite loop, the method will keep on calling itself if there is no condition to stop. So the question is, will this cycle go on infinitely? Of course, not. Your PC will crash, right? There is a way of controlling how many times the recursion occurs. We shall cover this later in this chapter.

Recursion is a way of problem solving by breaking down a broader problem into small parts. It is mostly used as an alternative to loop especially when using loop can be quite difficult. Recursion follows a bottom to top then top to bottom approach (or can be considered vise versa as per convenience).

13.2. Loop vs Recursion

Loop	Recursion
Not much difference in terms of use. Both can be used to solve repetitive problems.	
Constructs like for, while used.	Method calls itself and consists of a base case and a recursive case.
Loop condition given to stop loop execution at a point.	Stop executing when the base case is reached.
May not be easy to write about some complex problems.	May be easier to write complex problems and have more clarity.
Is more memory efficient.	Possibility of using more memory and resources due to stack overflow error.

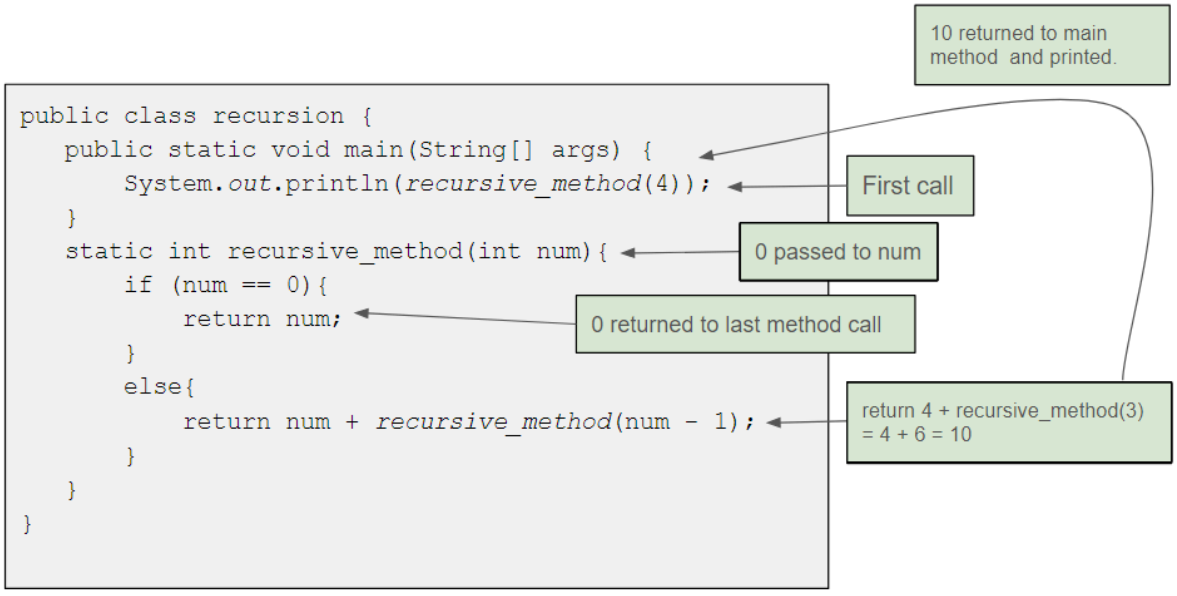
13.3. Components of Recursive Method

- **Base Case:** Termination condition of the recursion process.
- **Recursive Case:** Other cases other than the base case where the method keeps calling itself.

Can you tell what could be the issue if there was no base case? Look at the following example and try to understand.

```
static return_type recursive_method(datatype parameter){
    if (base case condition is satisfied){
        return final_value;
    }
    else{              //recursive case
        recursive_method(argument);
    }
}
```

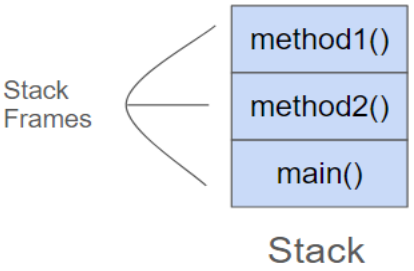
13.4. Method Calling with Recursion



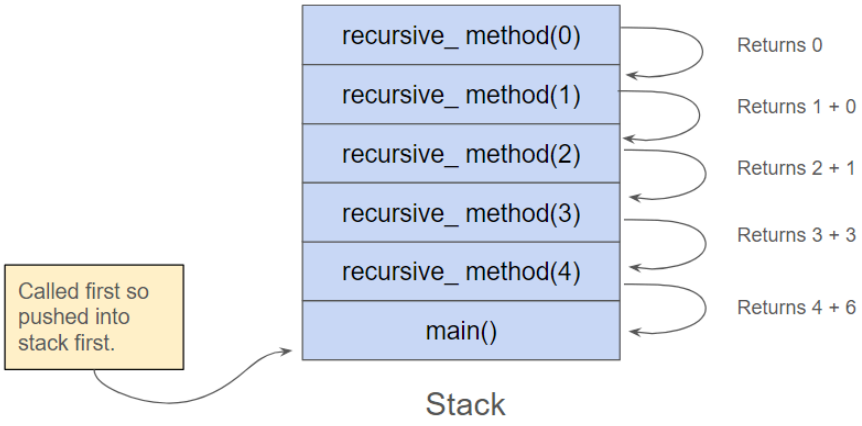
Here in the first method call of recursive_method, 4 was passed down as the parameter. After that a few successive recursive calls were made by the recursive_method and finally when the base case was satisfied (when num==0), the recursive_method ended and the final result was returned to the main method where it was printed.

13.5. Method Call Stack

Each method, when called, gets allocated into the stack frame. Each stack frame consists of the information about the called method like its local variables, arguments passed, returned value etc. A call stack example shown on the right. For a recursive method, every method call is also allocated a stack frame.



In the example shown in 13.4 the stack would look like this:

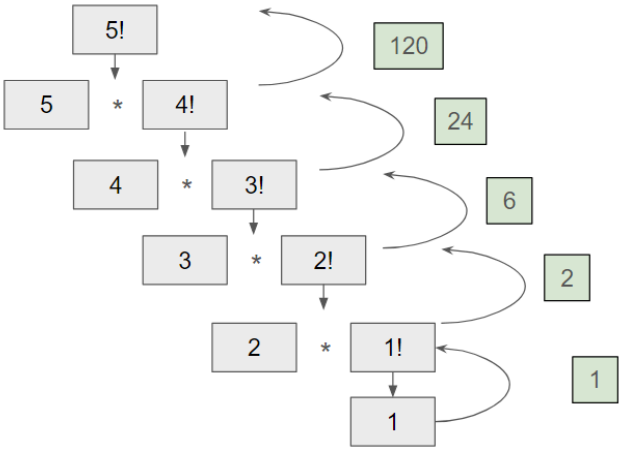


13.6. Recursion Example: Factorial

Loop Based Approach	Recursive Approach

`public class factorial {`
`  public static void main(String[] args) {`
`  int fact = 1;`
`  int number = 5;`
`  for (int i = 1; i<=5; i++){`
`  fact *= i;`
`  }`
`  System.out.println(fact);`
`  }`
`}`
	`public class factorial {`
	`  public static void main(String[] args) {`
	`  System.out.println(factorial_calculate(5));`
	`  }`
	`  static int factorial_calculate(int n){`
	`  if (n != 0) {`
	`  return n * factorial_calculate( n - 1);`
	`  }`
	`  else{`
	`  return 1;`
	`  }`
	`  }`
	`}`

Factorial of a number can be broken down into smaller parts.



13.7. Worksheet

A. Write a recursive method for printing the minimum value in an array.

```
public class Task1{
   public static void main(String[] args) {
        System.out.println(min({1,3,4,-3));
   }
   static int min(int n){
        //Your code here
   }
}
```

```
Output:
-3
```

B. Write a recursive method for finding the sum of the first n numbers in a Fibonacci series.

```
public class Task2{
   public static void main(String[] args) {
        System.out.println(fibonacci(5));
   }
   static int fibonacci(int n){
        //Your code here
   }
}
```

```
Output:
6
```


C. Write a recursive method that returns the sum of all the elements of an array.

```
public class Task3{
   public static void main(String[] args) {
        System.out.println(sum({1,3,4,-3));
   }
   static int sum(int n){
        //Your code here
   }
}
```

```
Output:
5
```