[In Class] Lesson 9 (10/30)

Meet Recursive Programming

In the pre-class work you were introduced to a new method of breaking down complex problems into a series of small steps a computer can do. Up until now we have always used **loops** or **"iteration"** to repeat lines of code, **recursion** is a different approach that can accomplish the same act of repeating lines of code, but instead of using a **loop** you will write methods that call themselves.

Let's consider the act of finding a "factorial". In mathematics, the factorial of a non-negative integer n, denoted by `n!`, is the product of all positive integers less than or equal to n. Ex: `4! = 1 * 2 * 3 * 4 = 24`

Parts of a Recursive Program

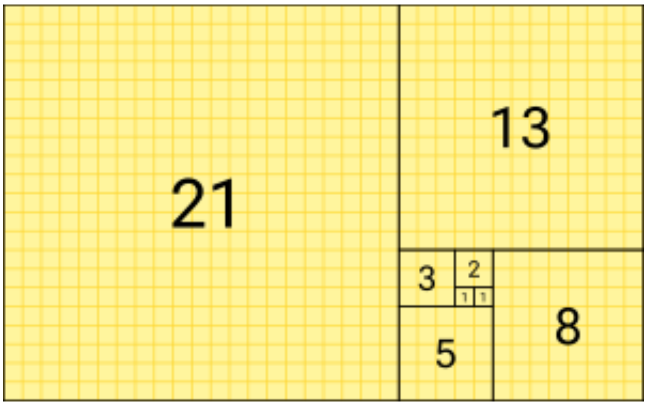
In the last example we saw that we could program Factorial by calling a method within itself, but we had to change the parameters with each call. The initial parameter passed in triggered a new call to the method, which triggered a new call and so on until the parameter was in a state that the **base case** was triggered instead of the **recursive case**.

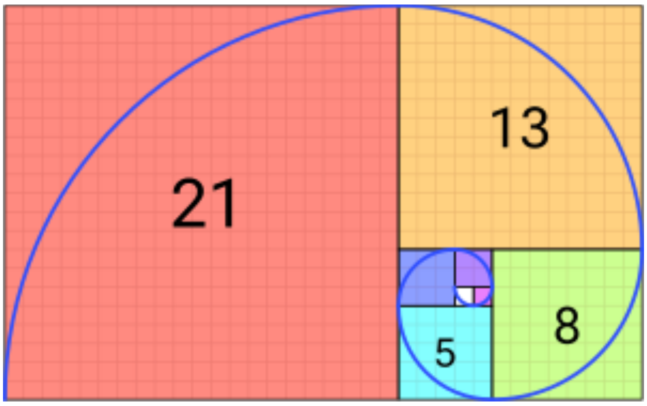
The **base case** is portion of the method that will not make another recursive call, it should be triggered by a specific state of the parameter that requires no more calculation. You can think of this like the recursive version of the for loop test, this case stops the recursive pattern. For Factorial our base case was 1 because the Factorial of 1 is 1, so there's nothing more to calculate.

The **recursive case** is the portion of the method that makes another recursive call, it should make a call to itself but must **CHANGE THE PARAMETERS TO TREND CLOSER TO THE BASE CASE.** In our Factorial example, this meant reducing the parameter by 1, because our base case was when n == 1 by subtracting 1 at each recursive call we are guaranteed to eventually hit the base case and stop the recursion.

Let's consider another classic recursive math problem: Fibonacci numbers. In mathematics, the Fibonacci numbers, F(n) , form a sequence, the Fibonacci sequence, in which each number is the sum of the two preceding ones. F(0) = 0 and F(1) = 1 which will act as our base cases. For example: F(4) = F(3) + F(2) = (F(2) + F(1)) + (F(1) + F(0)) = (((F(1) + F(0)) + F(1) + (F(1) + F(0))) = 1 + 0 + 1 + 1 + 0 = 3

$$F_n = F_{n-1} + F_{n-2}$$





Consider the recursive code shown. How many method calls will be generated by a call of `fib(5)`?

Recursive Mystery

The following question will give you practice in tracing through recursive code.

Consider the following method:

```
public void mystery1(int n) {
    if (n <= 1) {
        System.out.print(n);
    } else {
        mystery1(n / 2);
        System.out.print(", " + n);
    }
}
```

Question 1

What will be printed by `mystery1(1)` ?

No response

Question 2

What will be printed by `mystery1(4)` ?

No response

Question 3

What will be printed by `mystery1(16)` ?

No response

Question 4

What will be printed by `mystery1(30)` ?

No response

Recursive Mystery

The following question will give you practice in tracing through recursive code.

Consider the following method:

```
public void mystery3(int n) {
    if (n <= 0) {
        System.out.print("*");
    } else if (n % 2 == 0) {
        System.out.print("(");
        mystery3(n - 1);
        System.out.print(")");
    } else {
        System.out.print("[");
        mystery3(n - 1);
        System.out.print("]");
    }
}
```

Question 1

What will be printed by `mystery3(0)`?

No response

Question 2

What will be printed by `mystery3(1)`?

No response

Question 3

What will be printed by `mystery3(2)`?

No response

Question 4

What will be printed by `mystery3(4)`?

No response

Question 5

What will be printed by `mystery3(9)`?

No response