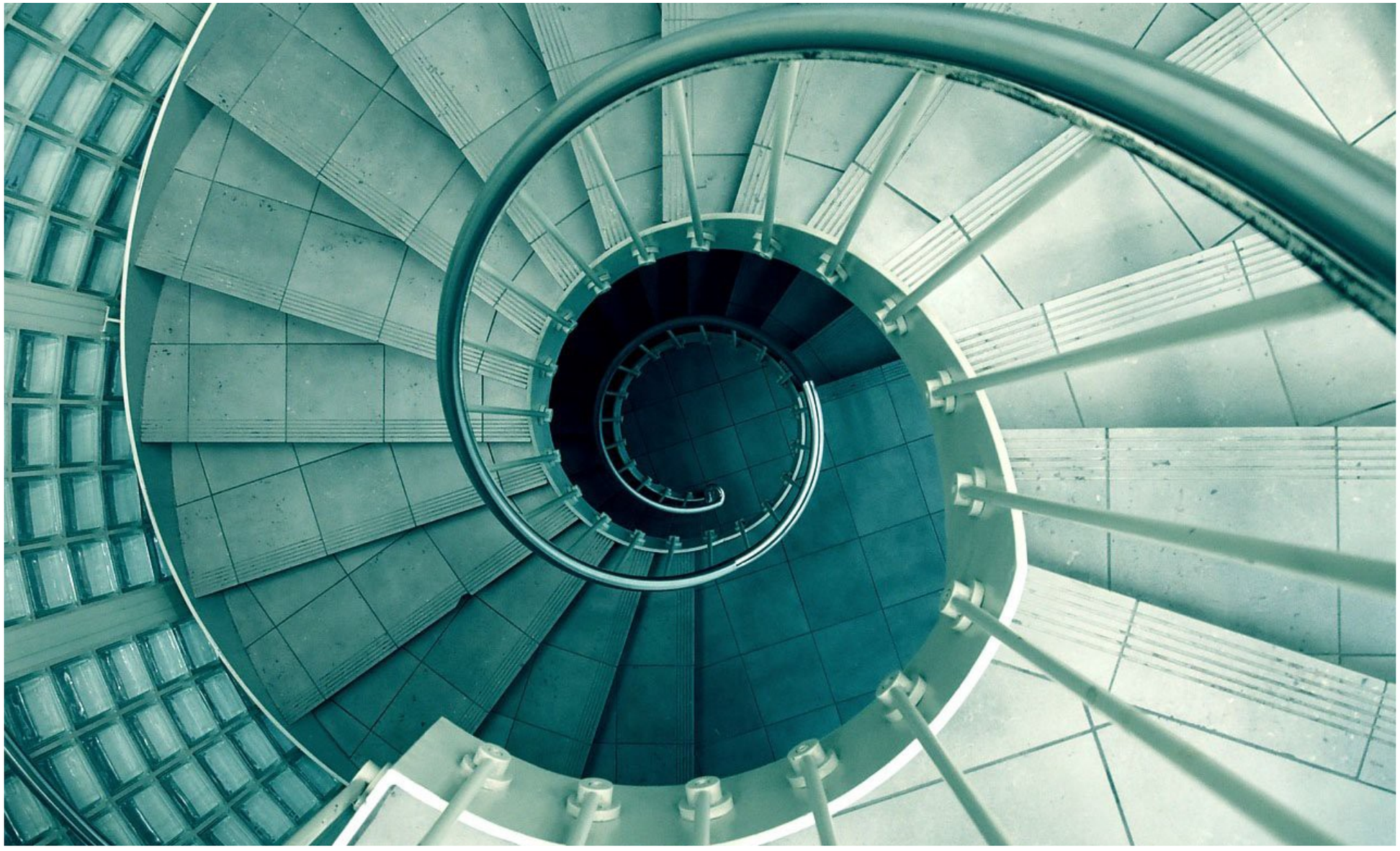


Java Recursion Method

2019 Lecture 4

Normal method

```java
public void normalMethod(int a, int b) {
    int c = a + b;
    int d = a * b;

    System.out.println(c);
    System.out.println(d);
}
```

Method with a return type
e.g. given n, return sum of 0 to n

```
public int normalMethod(int n) {
   int sum = 0;
   for (int i = n; i >=0; i--) {
      sum += i;
   }
   return sum;
}
```

Recursion Function

- Function will recursively calling itself
- Can have or not having a return type
- If not careful, can turn into infinite recursion

```java
public void recurMethod(int a) {
    System.out.println(a);
    recurMethod(a + 1);
}
```

```java
public int recurMethod(int a) {
    return recurMethod(a  - 1);
}
```

Proper designed Recursion Function

- Recursion function are usually used to solve problem that answers are depends on the previous calculation result.

- Recursion function should very carefully defined the BASE case, for function to stop

- Then carefully develop the recursion condition

Fibonacci sequence

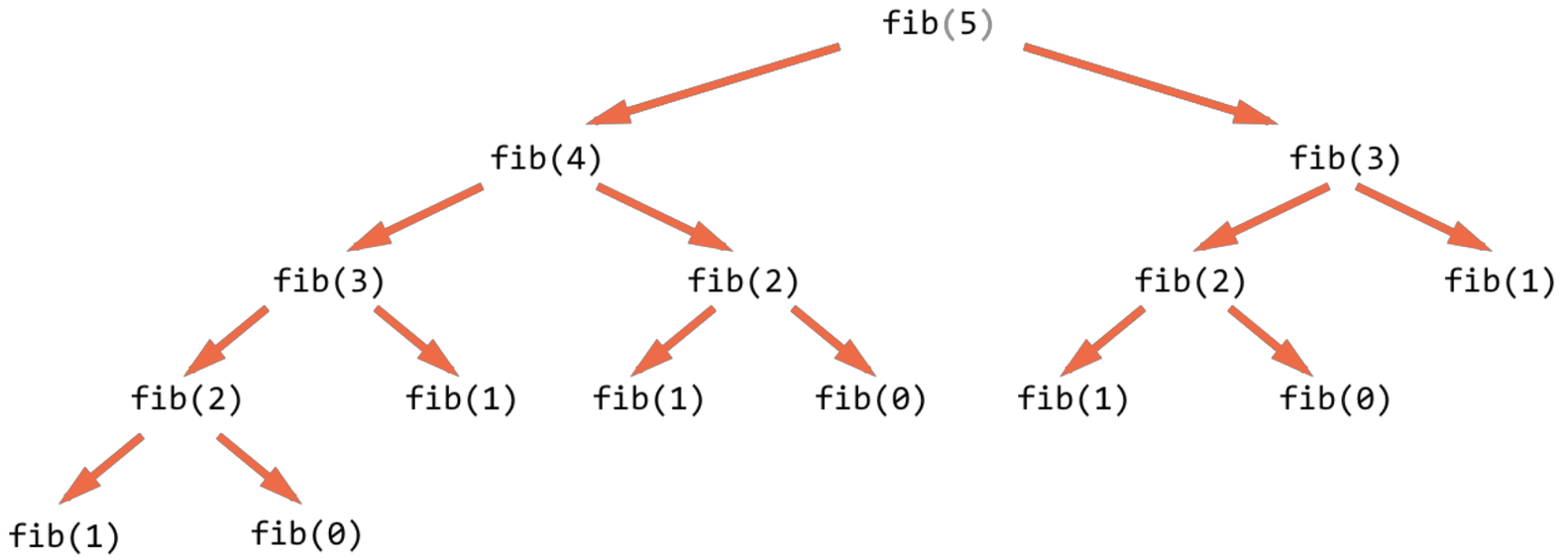
0 1 1 2 3 5 8 13.....

fib(n) = fib(n - 1) + fib(n -2)

Given int a, a is a number >=0, return the ath fibonacci number

```
public int fib(int a) {

}
```


fib(5)
fib(4)
fib(3)
fib(3)
fib(2)
fib(2)
fib(1)
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)
fib(1)
fib(0)

N!

N sum

```java
public int factor(int n) {
    if (n == 0) return 1;
    return factor(n - 1) * n;
}
```

```java
public int sum(int n) {
    if (n == 0) return 0;
    return sum(n - 1) + n;
}
```

Binary Search

```
public boolean binarySearch(int[] a, int key)
{
    int lo = 0;
    int hi = a.length - 1;
    while (lo <= hi)
    {
        // Key is in a[lo..hi] or not present.
        int mid = lo + (hi - lo) / 2; // do this to avoid overflow
        if (key < a[mid])
        {
            hi = mid - 1;
        }
        else if (key > a[mid])
        {

            lo = mid + 1;
        }
        else
        {
            return true;
        }
    }
    return false;
}
```

```java
public boolean binarySearch(int[ ] data, int target, int low, int high) {
    if (low > high) {
        return false;
    } else {
        int mid = low + (high - low) / 2; // do this to avoid overflow
        if (target == data[mid])
            return true;
        else if (target < data[mid])
            return binarySearch(data, target, low, mid - 1);
        else
            return binarySearch(data, target, mid + 1, high);
    }
}
```