What we will cover:
Recursion
Factorials
Fibonacci Numbers

Recursion
We have discussed the idea of decomposition quite a bit. Our solutions tend to take a large problem and break it up into small, solvable sub-problems, which we then solve individually. Sometimes, the pieces of code that we have created simply need to be called with different parameters to complete a large task. Imagine a deeply nested directory structure where the deepest directory must be deleted before any parent directories can be deleted. You could create a method that deletes a single directory, but for this case, you would need to discover the deepest directory at any time and call the delete method on that directory. This would also apply in cases of file search as well.

Recursive methods are methods that call or invoke themselves, either directly, or indirectly. The code for these methods is usually short and very elegant. Often, however, they are not very efficient in terms of memory usage and computation time.

Factorials
Recall, that mathematically, factorial is defined as:

\[ n! = \prod_{k=1}^{n} k \]

Or recursively defined as:

\[ n! = \begin{cases} 
1, & \text{if } n = 0 \\
n \cdot (n-1)!, & \text{if } n > 0 
\end{cases} \]

We showed an iterative solution for this problem earlier this year. We can also solve this problem recursively. Notice that the definition of factorial states that the factorial of a number n is the number n times the factorial of n-1. This type of problem lends itself towards a recursive solution.

All recursive solutions must have what is known as a base case. This is sometimes called a stopping condition and this is always defined as the simplest case of the problem. Since a recursive method works by calling itself on smaller and smaller pieces of a problem, it must also know when to stop to avoid working forever. Luckily, the factorial definition provides our base case, 0! = 1. We know that once we reach zero, we are at the simplest case and can return a result.

Our recursive method solution for this particular problem would look like this:

```java
public static long factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Does this code do what we need it to do? Is there a base case defined? What is it?

Class Exercise 46
Fibonacci Numbers

Another well-known series of numbers are the Fibonacci numbers. This series recurrence relation can be written as:

\[ f_{ib}(n) = \begin{cases} 
0, & \text{if } n = 0 \\
1, & \text{if } n = 1 \\
f_{ib}(n - 1) + f_{ib}(n - 2), & \text{if } n > 1 
\end{cases} \]

This series is represented numerically as 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

The Fibonacci series appears all over nature and in design. The golden spiral, a spiral created using squares in a Fibonacci tiling, is one of the more common representations. Examples in nature include tree branching, leaf arrangement on stems, pineapple sprouts and many others\(^1\).

We can create recursive code for this series in the same manner as we did for the factorial. First, define your base cases so that the recursive calls will end at some point, then set up your recursive calls to the function.

\[ \text{Class Exercise 47} \]

\[1 \text{ http://britton.disted.camosun.bc.ca/fibslide/jbfibslide.htm} \text{ as one example. Many other resources are available.} \]