12.1 – Recursive Thinking

- **Recursion** is a programming technique in which a method can call itself in order to fulfill its purpose.
- A recursive definition is one which uses the word or concept being defined in the definition itself.
  - A recursive definition must have a non-recursive part. If not, there is no way to terminate the recursive path.
- All recursive definitions must have a non-recursive part. A definition without a non-recursive part causes infinite recursion.
- The non-recursive part is called the base case.
- In some situations, a recursive definition can be an appropriate way to express a concept.

12.2 – Recursive Definitions in Math

- Mathematical formulas are often expressed recursively.
- You can define N!, for any positive integer N, as:
  \[ N! = N \times (N-1) \times \ldots \times 2 \times 1 \]
- This definition can be expressed recursively:
  \[ 1! = 1 \]
  \[ N! = N \times (N-1)! \]

12.2 – Recursive Programming

- A method in Java can invoke itself (and is called a recursive method).
- The code of a recursive method must be structured to handle both the base case and the recursive case.
- Each call sets up a new execution environment, with new parameters and new local variables.

```java
public int prod(int num) {
    int result = 1;
    if (num == 1) { // Base case
        result = 1;
    } else { // Recursive case
        result = num * prod(num-1);
    }
    return result;
}
```

### Recursive calls to the `prod` method

```
public int prod(int num) {
    int result;
    if (num == 1) { // Base case
        result = 1;
    } else { // Recursive case
        result = num * prod(num-1);
    }
    return result;
}
```
12.2 – Recursion vs. Iteration

- Every recursive solution has a corresponding iterative solution.
- For example, $N!$ can also be calculated with a loop.
- Recursion has the overhead of multiple method invocations.
- However, for some problems, recursive solutions are often more simple and elegant than iterative solutions.

12.3 – The Towers of Hanoi puzzle

- The Towers of Hanoi is a puzzle made up of three vertical pegs and several disks of increasing size that slide onto the pegs.
- The goal is to move all of the disks from one peg to another following these rules:
  - Only one disk can be moved at a time.
  - A disk cannot be placed on top of a smaller disk.
  - All disks must be on some peg (except for the one in transit).

12.3 – A solution to the 3-disk Hanoi

To move a stack of $N$ disks from the original peg to the destination peg:
- Move the topmost $N-1$ disks from the original peg to the extra peg.
- Move the largest disk from the original peg to the destination peg.
- Move the $N-1$ disks from the extra peg to the destination peg.
- The base case occurs when a “stack” contains only one disk.

- The recursive solution is simple and elegant to express (and program).
- An iterative solution to this problem is much more complex.
- Note that the number of moves increases exponentially as the number of disks increases.
12.3 – SolveTowers.java

```java
//****************************************************************
//  SolveTowers.java       Java Foundations
//  Demonstrates recursion by solving the Towers of Hanoi puzzle.
//****************************************************************
public class SolveTowers
{
    public static void main (String[] args)
    {
        TowersOfHanoi towers = new TowersOfHanoi (4);
        towers.solve();
    }
}
```