Recursion

**Recursive Definition** – “An ancestor is a parent or an ancestor of a parent”

* Consists of something we are familiar with (a parent), and some sort of *recurrence relation* involving:
* The word we are trying to define & the familiar entity (ancestor of a parent)

\*Strength: the *recurrence relation* (the *repetitive* part of the definition) is used to briefly & clearly expand the scope of the definition

**Recursive Algorithm** –

* An algorithm defined at least *partially* in terms of itself
* Not all problems have recursive solutions
* The ones that do – provides an alternative to iterative algorithms (involving loops)

**Calculating n Factorial** –

* For n = 0, n! = 1
* For all (+) values of n, n! = n \* (n – 1) \* (n – 2) \* (n – 3) \* … \* 1
* 0! = 1 (by definition)

Coding The Algorithm 🡪 Tracing Its Execution Path –

* \*Any Java Method can implement a Recursive Algorithm\*
* One indication that a method implements a recursive algorithm 🡪 it invokes itself
* Directly Recursive: method invokes itself
* Indirectly Recursive: method invokes some other method that invokes it

**Formulating a Recursive Method** –

* Techniques for formulating Recursive Algorithms can be methodized
* Understanding of these methodized approaches:
* Further understanding of recursion
* Can take you far into the development of most recursive algorithms
* Reveals insights into the thought processes of those w/ the innate ability to think recursively

Definitions:

* **Base Case**:
* The known portion of the problem solution
* The nonrecursive portion of the solution (part “everyone knows”)
* The *Escape Clause* that allows the algorithm to escape from the recursive invocations
* **0! = 1**
* **Reduced Problem**:
* A problem very “close” to the original problem, but slightly closer to the base case
* When it is repeatedly reduced – it degenerates into the Base Case
* \*If it doesn’t eventually degenerate into the base class – the algorithm will enter an infinite loop 🡪 false base case
* **(n – 1)!**
* **General Solution**:
* The solution to the original problem (n!) expressed in terms of the reduced problem
* Uses the solution to the reduced problem to solve the original problem
* **n \* (n – 1)!**

n!

Triangular Numbers:

* the number of objects needed to form an equilateral triangle