# **Recursion (Part 1)**



EECS2101 X & Z: Fundamentals of Data Structures Winter 2025

CHEN-WEI WANG



- Fantastic resources for developing your recursive skills: http://codingbat.com/java/Recursion-1 http://codingbat.com/java/Recursion-2
- The best long-term approaches for mastering recursion are:

   learning a functional programming language
   [e.g., Haskell: https://www.haskell.org/tutorial/]
   learning to develop a compiler (after learning trees in this course)
   [e.g., ANTLR4 from EECS4302]



# **Background Study: Basic Recursion**

- It is assumed that, in EECS2030, you learned about the basics of recursion in Java:
  - What makes a method recursive?
  - How to trace recursion using a *call stack*?
  - How to define and use *recursive helper methods* on arrays?
- If needed, review the above assumed basics from the relevant parts of EECS2030:
  - From F'21: Parts A C, Lecture 8, Week 12
  - From F'24: Lecture 24, Sec. E (Tower of Hanoi)

Tips.

- Skim the *slides*: watch lecture videos if needing explanations.
- Recursion lab from EECS2030-F22: here [Solution: here]
- Ask questions related to the assumed basics of *recursion*!
- Assuming that you know the basics of *recursion*, we will:
  - Look at an advanced example of *recursion on arrays* together.
  - Have you complete an assignment on the more advanced recursion problems.



This module is designed to help you:

- Quickly review the *recursion basics*.
- Know about the <u>resources</u> on *recursion basics*.
- Get used to the more advanced use of recursion.



- Recursion is useful in expressing solutions to problems that can be recursively defined:
  - Base Cases: Small problem instances immediately solvable.
  - Recursive Cases:
    - Large problem instances not immediately solvable.
    - Solve by reusing *solution(s)* to *strictly smaller* problem instances.
- Similar idea learnt in high school: [ mathematical induction ]

### Tracing Method Calls via a Stack



- When a method is called, it is *activated* (and becomes *active*) and *pushed* onto the stack.
- When the body of a method makes a (helper) method call, that (helper) method is *activated* (and becomes *active*) and *pushed* onto the stack.
  - $\Rightarrow$  The stack contains activation records of all *active* methods.
  - Top of stack denotes the current point of execution .
  - Remaining parts of stack are (temporarily) suspended.
- When entire body of a method is executed, stack is *popped*.
  - ⇒ The current point of execution is returned to the new top of stack (which was suspended and just became active).
- Execution terminates when the stack becomes empty.

# Tracing Method Calls via a Stack



• Can you identify the pattern of a Fibonacci sequence?

 $F = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$ 

• Here is the formal, *recursive* definition of calculating the *n*<sub>th</sub> number in a Fibonacci sequence (denoted as *F*<sub>n</sub>):

$$F_n = \begin{cases} 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ F_{n-1} + F_{n-2} & \text{if } n > 2 \end{cases}$$

- · Your tasks are then to review how to
  - *implement* the above mathematical, recursive function in Java
  - trace, via a stack, the recursive execution at runtime

by studying **this video** (≈ 20 minutes):

7 of 10

### Making Recursive Calls on an Array



- For *efficiency*, we exploit the feature of *call by value*, by:
  - passing the *reference* of the same array
  - specifying the range of indices to be considered

```
void m(int[] a, int from, int to) {
    if(from > to) { /* base case */ }
    else if(from == to) { /* base case */ }
    else { m(a, from + 1, to) } }
```

- m(a, 0, a.length 1) [Initial call; entire array]
- m(a, 1, a.length 1) [1st r.c. on array of size a.length 1]
- m(a, a.length-1, a.length-1) [Last r.c. on array of size 1]

#### • Required Task:

Study the two examples allPositive and isSorted from the background study.



**Assuming** that you will review the assumed basic, let's go over an advanced example from paper to Eclipse:

### • Problem Description:

https://www.eecs.yorku.ca/~wangcw/teaching/ lectures/2025/W/EECS2101/exercises/ EECS2101-W25-Problem-Recursion-splitArray-Spec.pdf

### • Java starter project (with hints and JUnit tests):

https://www.eecs.yorku.ca/~wangcw/teaching/ lectures/2025/W/EECS2101/exercises/ ExtraRecursionProblemSplitArray\_Starter.zip

# Index (1)



Beyond this lecture ....

**Background Study: Basic Recursion** 

Learning Outcomes of this Lecture

**Recursion: Principle** 

Tracing Method Calls via a Stack

Tracing Method Calls via a Stack

Making Recursive Calls on an Array

A More Advanced Example on Recursion