Recursive Programming

- Consider the problem of computing the sum of all the numbers between 1 and any positive integer $N$.
- This problem can be recursively defined as:

$$\sum_{i=1}^{N} i = N + \sum_{j=1}^{N-1} j$$

$$= N + (N-1) + \sum_{j=1}^{N-2} j$$

$$= N + (N-1) + (N-2) + \sum_{j=1}^{N-3} j$$

Recursive Programming

```java
// This method returns the sum of 1 to num
public int sum(int num) {
    int result;
    if (num == 1) {
        result = 1;
    } else {
        result = num + sum(num-1);
    }
    return result;
}
```

Recursive Control Flow

- In Recursive calls methods can call themselves, but typically with different arguments each time.

Stack Animation

Memory Organization

The stack during a recursive call to gcd

Towers of Hanoi

- The Towers of Hanoi is a puzzle made up of three vertical pegs and several disks that slide on the pegs.
- The disks are of varying size, initially placed on one peg with the largest disk on the bottom with increasingly smaller ones on top.
- The goal is to move all of the disks from one peg to another under the following rules:
  - We can move only one disk at a time.
  - We cannot move a larger disk on top of a smaller one.
Animation of the Towers of Hanoi


Mystery Recursion on HW8

```java
public static void mystery1(int a, int b)
{
    if (a <= b) {
        int m = (a + b) / 2;
        System.out.print(m + " ");
        mystery1(a, m-1);
        mystery1(m+1, b);
    }
}

public static void main(String[] args) {
    mystery1(0, 5);
    System.out.println();
}
```

Think of recursion as a tree …

... an upside down tree
Recursion: Fibonacci Numbers

The sequence: \(\{0, 1, 1, 2, 3, 5, 8, 13, \ldots\}\)

\[
F_n = \begin{cases} 
0, & n = 0 \\
1, & n = 1 \\
F_{n-1} + F_{n-2}, & n \geq 2 
\end{cases}
\]

Mathematical notation v.s. java code

\[
F_n = \begin{cases} 
0, & n = 0 \\
1, & n = 1 \\
F_{n-1} + F_{n-2}, & n \geq 2 
\end{cases}
\]

```java
public static int fib(int n) {
    if(n <= 1) return n; // base case
    else return fib(n-1) + fib(n-2);
}
```
Execution Trace

Indirect Recursion
- A method invoking itself is considered to be direct recursion
- A method could invoke another method, which invokes another, etc., until eventually the original method is invoked again
- For example, method m1 could invoke m2, which invokes m3, which in turn invokes m1 again
- This is called indirect recursion, and requires all the same care as direct recursion
- It is often more difficult to trace and debug

Maze Traversal
- We can use recursion to find a path through a maze
- From each location, we can search in each direction
- Recursion keeps track of the path through the maze
- The base case is an invalid move or reaching the final destination
- See MazeSearch.java (page 583)
- See Maze.java (page 584)