Factorial!

```
public class FactorialCalculator {

    // Recursive method to calculate factorial
    public static int factorial(int n) {
        // Base case: if n is 1, simply return 1
        if (n == 1) {
            return 1;
        } else {
            // Recursive case: n times factorial of n-1
            return n * factorial(n - 1);
        }
    }

    public static void main(String[] args) {
        int number = 5; // Example number
        int result = factorial(number);
        System.out.println("Factorial of " + number + " is " + result);
    }
}
```

Fibonacci Calculator

```
public class FibonacciCalculator {

    // Recursive method to calculate Fibonacci number
    public static int fibonacci(int n) {
        // Base cases
        if (n == 0) {
            return 0;
        }
        if (n == 1) {
            return 1;
        }
        // Recursive case
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    public static void main(String[] args) {
        int index = 10; // Example index
        int result = fibonacci(index);
        System.out.println("Fibonacci number at index " + index + " is " + result);
    }
}
```

Output

```
fibonacci(10)
  fibonacci(9)
    fibonacci(8)
      fibonacci(7)
        fibonacci(6)
          fibonacci(5)
            fibonacci(4)
              fibonacci(3)
                fibonacci(2)
                  fibonacci(1) -> returns 1
                  fibonacci(0) -> returns 0
                fibonacci(1) -> returns 1
              fibonacci(2) -> returns 1
            fibonacci(3) -> returns 2
          fibonacci(4) -> returns 3
        fibonacci(5) -> returns 5
      fibonacci(6) -> returns 8
    fibonacci(7) -> returns 13
  fibonacci(8)
    (Already calculated above, reuse the value -> returns 21)
  fibonacci(9) -> returns 34
fibonacci(10) -> returns 55
```

Tower of Hanoi

```
function moveDisks(n, sourcePole, destinationPole, auxiliaryPole)
    if n == 1 then
        print "Move disk from " + sourcePole + " to " + destinationPole
    else
        moveDisks(n-1, sourcePole, auxiliaryPole, destinationPole)
        print "Move disk from " + sourcePole + " to " + destinationPole
        moveDisks(n-1, auxiliaryPole, destinationPole, sourcePole)
end function
```

Detailed Trace for n = 3:

Initial Setup:

- Disks: 3, labeled from top as Disk 1, Disk 2, and Disk 3 (Disk 3 is the largest)
- Source Pole (A)
- Destination Pole (C)
- Auxiliary Pole (B)

Function Call:

- moveDisks(3, A, C, B)

Trace Explanation:

1. First Recursive Call: moveDisks(2, A, B, C)
    - Objective: Move the top 2 disks from A to B using C as the destination temporarily.

    Exploring moveDisks(2, A, B, C):

    2. Recursive Call: moveDisks(1, A, C, B)
        - Objective: Move Disk 1 from A to C using B as temporary storage.
        - Action: Moves Disk 1 directly from A to C.
        - Print: "Move disk from A to C"
    3. Move Disk 2 Directly: After moving Disk 1 to C, move Disk 2 from A to B.
        - Print: "Move disk from A to B"
    4. Recursive Call: moveDisks(1, C, B, A)
        - Objective: Move Disk 1 from C to B using A as temporary storage.
        - Action: Moves Disk 1 directly from C to B.
        - Print: "Move disk from C to B"
2. Direct Move of Disk 3: With Disks 1 and 2 on B, move Disk 3 from A to C.
    - Print: "Move disk from A to C"
3. Second Recursive Call: moveDisks(2, B, C, A)
    - Objective: Move the 2 disks from B to C using A as temporary storage.

    Exploring moveDisks(2, B, C, A):

    1. Recursive Call: moveDisks(1, B, A, C)
        - Objective: Move Disk 1 from B to A using C as temporary storage.
        - Action: Moves Disk 1 directly from B to A.
        - Print: "Move disk from B to A"
    2. Move Disk 2 Directly: Move Disk 2 from B to C.
        - Print: "Move disk from B to C"
    3. Recursive Call: moveDisks(1, A, C, B)
        - Objective: Move Disk 1 from A to C using B as temporary storage.
        - Action: Moves Disk 1 directly from A to C.
        - Print: "Move disk from A to C"

Tail Recursive

```
public class TailRecursiveFactorial {

    public static int factorial(int n) {
        return factorialTailRec(n, 1);  // Start the recursive chain with 1 as the initial accumulator value
    }

    private static int factorialTailRec(int current, int accumulator) {
        if (current == 0) {
            return accumulator;  // Return the accumulated value when reaching the base case
        }
        return factorialTailRec(current - 1, accumulator * current);  // Pass the result of the multiplication to the next recursive call
    }

    public static void main(String[] args) {
        int result = factorial(5);
        System.out.println("Tail Recursive Factorial of 5 is " + result);
    }
}
```

Indirect Recursion

```
public class IndirectRecursionExample {

    public static void funcA(int n) {
        if (n <= 0) {
            System.out.println("Reached the base case in funcA");
            return;
        }
        System.out.println("funcA: " + n);
        funcB(n - 1);
    }

    public static void funcB(int n) {
        if (n <= 0) {
            System.out.println("Reached the base case in funcB");
            return;
        }
        System.out.println("funcB: " + n);
        funcA(n - 2); // Decrements by 2 to add variety to the countdown and show different progression.
    }

    public static void main(String[] args) {
        funcA(10); // Start the indirect recursion with funcA
    }
}
```