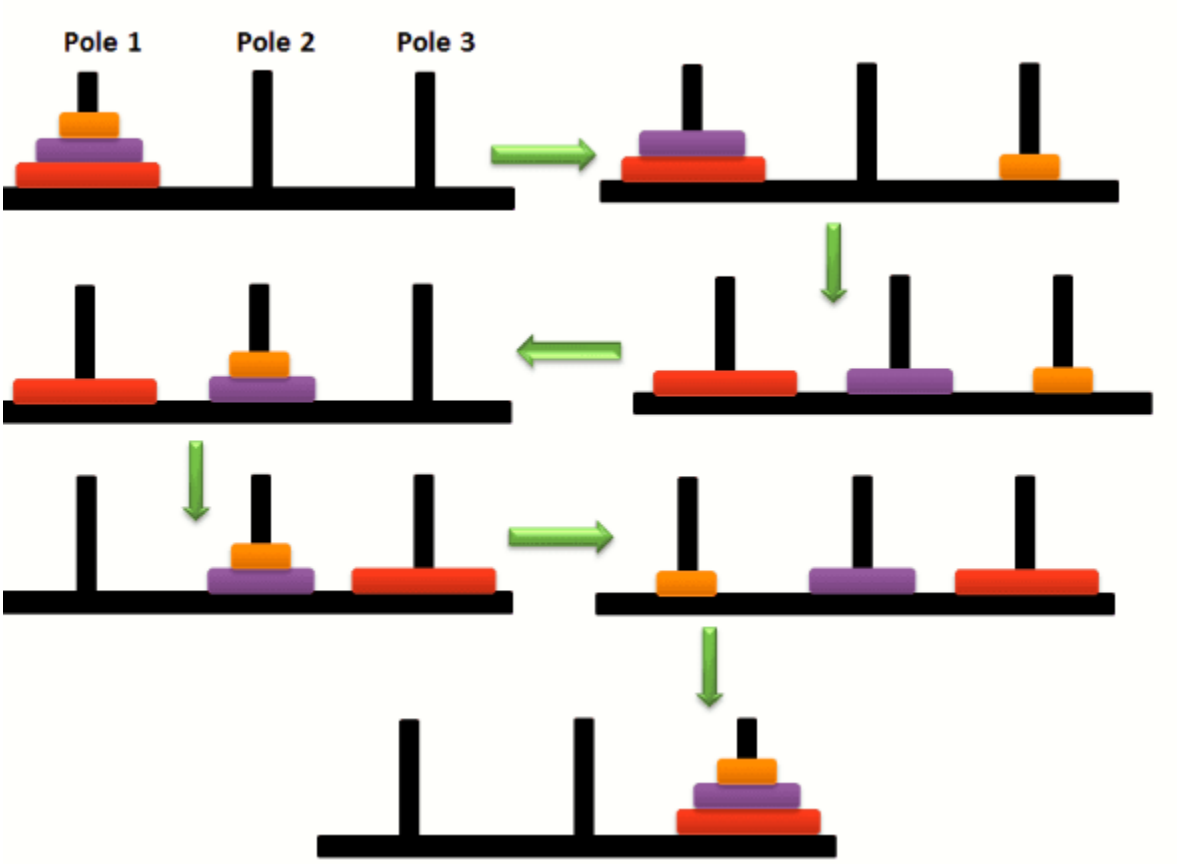
Module 9 Assignment – Recursion

Instructions

In this assignment, you will copy the code for the classic recursion demo project, Towers of Hanoi.

Tip: when you run the program, don't make the input too large.

Here is an infographic of the steps involved:



Read the definition of recursion in programming on the next page.

After that, you will see some images of the two files you implement:

Towers.java

Runner.java

After your files run, you will fire up your debugger and step through the code. Take at least 3 screenshots of your variables.

In programming, recursion is a technique where a function calls itself to solve a problem by breaking it down into smaller, similar subproblems until a base case is reached, allowing the recursion to terminate.

Here's a more detailed explanation:

- **Function Calling Itself:**

A recursive function is defined as a function that calls itself within its own code.

- **Breaking Down Problems:**

Recursion is often used to solve problems that can be naturally divided into smaller, self-similar subproblems.

- **Base Case:**

Every recursive function must have a base case, which is a condition that stops the recursion from continuing indefinitely. Without a base case, the function would call itself endlessly, leading to a stack overflow error.

- **Recursive Step:**

The recursive step is where the function calls itself with a modified input, moving closer to the base case.

- **Example:**

A common example of recursion is calculating the factorial of a number. The factorial of a number n (denoted as n!) is the product of all positive integers from 1 to n. A recursive function to calculate the factorial would call itself with n-1 until n reaches 1 (the base case).

    - **Base Case:** factorial(1) = 1
    - **Recursive Step:** factorial(n) = n * factorial(n-1)

- **Advantages:**

Recursion can make code more concise and easier to read for certain problems, especially those involving tree-like structures or inherently recursive algorithms.

- **Disadvantages:**

Recursion can sometimes be less efficient than iterative solutions, especially for problems that can be solved iteratively, and it can lead to stack overflow errors if the recursion depth is too large.

Towers.java code:

```java
public class Towers {
    //Solution to the Towers ofHanoi game.
    private int numDiscs;   // Number of discs
    //Constructor param - the number of discs to use
    //Constructor calls a recursive method
    public Towers(int n) {
        this.numDiscs = n;

        // Move the number of discs from peg 1 to peg 3
        // using peg 2 as a temporary storage location.
        //calling recursive method
        this.moveDiscs(this.numDiscs, 1, 3, 2);
    }

    //The moveDiscs method displays a disc move using a recursive algorithm
    private void moveDiscs(int num, int fromPeg,
                           int toPeg, int tempPeg) {
        if (num > 0) {
            //recursive call - creates another whole copy of method in RAM
            moveDiscs(num - 1, fromPeg, tempPeg, toPeg);
            System.out.println("Move a disc from peg " +
                    fromPeg + " to peg " + toPeg);

            moveDiscs(num - 1, tempPeg, toPeg, fromPeg);
        }
    }
}
```

Runner.java code:

```java
public class Runner {
    ///Instantiate a Towers object to swap the rings on 3 "towers"
    public static void main(String[] args)
     {
            Towers towersOfHanoi = new Towers(3);
      }

}
```

In addition to you code, create a screenshot file in Word, and take at least 3 screenshots during the debugging process. Submit to GitHub.