towerOfHanoi.java

```java
/*
Iterative Solution of Tower of Hanoi using Stack
The Tower of Hanoi is a mathematical puzzle. It consists of three poles
and a number of disks of different sizes which can slide onto any poles.
The puzzle starts with the disk in a neat stack in ascending order of size
in one pole, the smallest at the top thus making a conical shape. The objective
of the puzzle is to move all the disks from one pole (say 'source pole') to another
pole (say 'destination pole') with the help of the third pole (say auxiliary pole).
The puzzle has the following two rules:
      1. You can't place a larger disk onto a smaller disk
      2. Only one disk can be moved at a time
 */
public class towerOfHanoi {
    public static void main(String[] args) {
        int disks = 3;
        towerOfHanoi toh = new towerOfHanoi();
        // Create three stacks of size disk to hold the disks
        Stack src = toh.create(disks);
        Stack dest = toh.create(disks);
        Stack aux = toh.create(disks);
        toh.solution(disks, src, aux, dest);
    }

    Stack create(int size) {
        Stack stack = new Stack();
        stack.size = size;
        stack.top = -1;
        stack.arr = new int[size];
        return stack;
    }

    boolean isFull(Stack stack) {
        return (stack.top == stack.size - 1);
    }

    boolean isEmpty(Stack stack) {
        return (stack.top == -1);
    }

    // Adding an item to the stack.
    void push(Stack stack, int n) {
        if (isFull(stack)) return;
        stack.arr[++stack.top] = n;
    }

    // Removing an item from the top
    int pop(Stack stack) {
        if (isEmpty(stack))
            return Integer.MIN_VALUE;
        return stack.arr[stack.top--];
    }

    void moveDisksBetweenPoles(Stack src, Stack dest, char s, char d) {
        int pole1TopDisk = pop(src);
        int pole2TopDisk = pop(dest);

        // When pole 1 is empty
        if (pole1TopDisk == Integer.MIN_VALUE) {
            push(src, pole2TopDisk);
            moveDisk(d, s, pole2TopDisk);
        }

        // When pole2 pole is empty
        else if (pole2TopDisk == Integer.MIN_VALUE) {
            push(dest, pole1TopDisk);
            moveDisk(s, d, pole1TopDisk);
        }

        // When top disk of pole1 > top disk of pole2
        else if (pole1TopDisk > pole2TopDisk) {
            push(src, pole1TopDisk);
            push(src, pole2TopDisk);
            moveDisk(d, s, pole2TopDisk);
        }
        // When top disk of pole1 < top disk of pole2
        else {
            push(dest, pole2TopDisk);
            push(dest, pole1TopDisk);
            moveDisk(s, d, pole1TopDisk);
        }
```

```java
    }

    void moveDisk(char fromRod, char toRod, int disk) {
        System.out.println("Move disk " + disk + " from " + fromRod + " to " + toRod);
    }

    void solution(int disks, Stack src, Stack aux, Stack dest) {
        int i, noOfMoves;
        char s = 'S', d = 'D', a = 'A';
        // If number of disks is even, then interchange destination pole and auxiliary pole
        if (disks % 2 == 0) {
            char tmp = d;
            d = a;
            a = tmp;
        }
        noOfMoves = (int) (Math.pow(2, disks) - 1);

        // Larger disks will be pushed first
        for (i = disks; i >= 1; i--) {
            push(src, i);
        }

        for (i = 1; i <= noOfMoves; i++) {
            if (i % 3 == 1) moveDisksBetweenPoles(src, dest, s, d);
            else if (i % 3 == 2) moveDisksBetweenPoles(src, aux, s, a);
            else if (i % 3 == 0) moveDisksBetweenPoles(aux, dest, a, d);
        }
    }

    class Stack {
        int size;
        int top;
        int[] arr;
    }
}
```