4.2 Tower of Hanoi

The Tower of Hanoi (or Towers of Hanoi) is a mathematical game. It consists of three rods and a number of disks of different sizes which can slide onto any rod. The game starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape (from Wikipedia, the free encyclopedia).

The objective is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

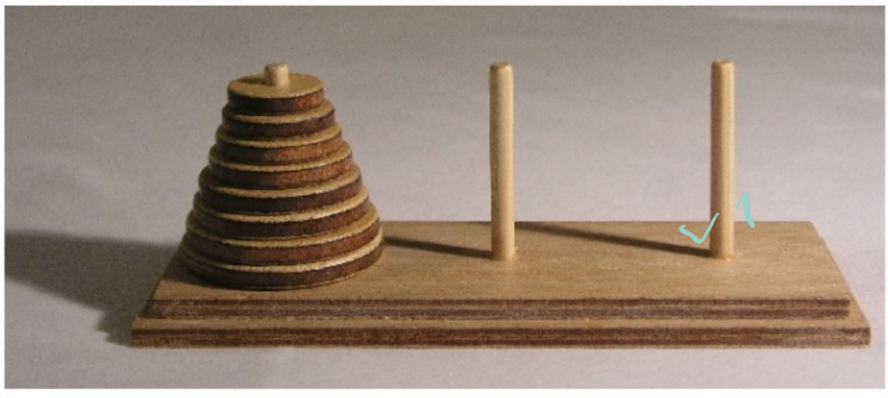

Figure 4.5: Tower of Hanoi

4.2.1 Recursive solution

A key to solving this puzzle is to recognize that it can be solved by breaking the problem down into a collection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. The following procedure demonstrates this approach.

- label the pegs A, B, C – these labels may move at different steps
- let n be the total number of discs
- number the discs from 1 (smallest, topmost) to n (largest, bottommost)

To move n discs from peg A to peg C:

1. Move $n-1$ discs from A to B. This leaves one disc alone on peg A.
2. Move this disc from A to C.
3. Move $n-1$ discs from B to C so they sit on the last disc.

To carry out steps 1 and 3, apply the same algorithm again for $n-1$. The entire procedure is a finite number of steps, since at some point the algorithm will be required for $n=1$. This step, moving a single disc from peg A to peg B, is trivial.

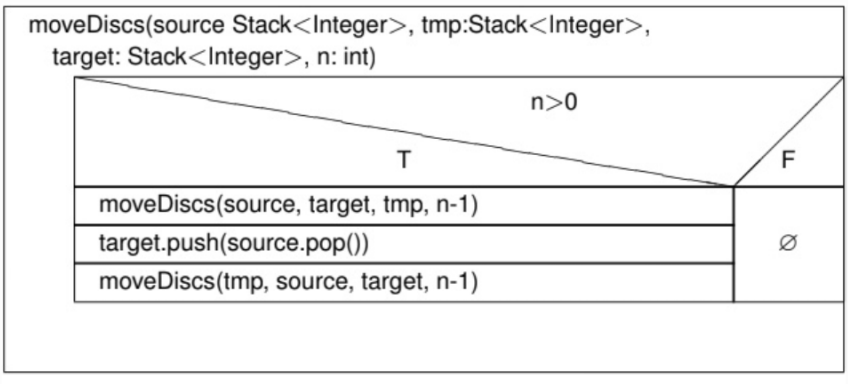

Figure 4.6: Recursive algorithm to move n discs

4.2.2 Nassi-Shneiderman diagram

4.2.3 Class design

Create a class for Tower of Hanoi objects. Use three stacks to model the pegs for the discs. Implement a constructor with the number of discs which have to be moved from the source peg to the target peg as parameter. The constructor creates the pegs and the discs. A disc of a certain size can be represented by an integer number. Implement the class for Tower of Hanoi objects using the following UML diagram and additional Nassi-Shneiderman diagrams.

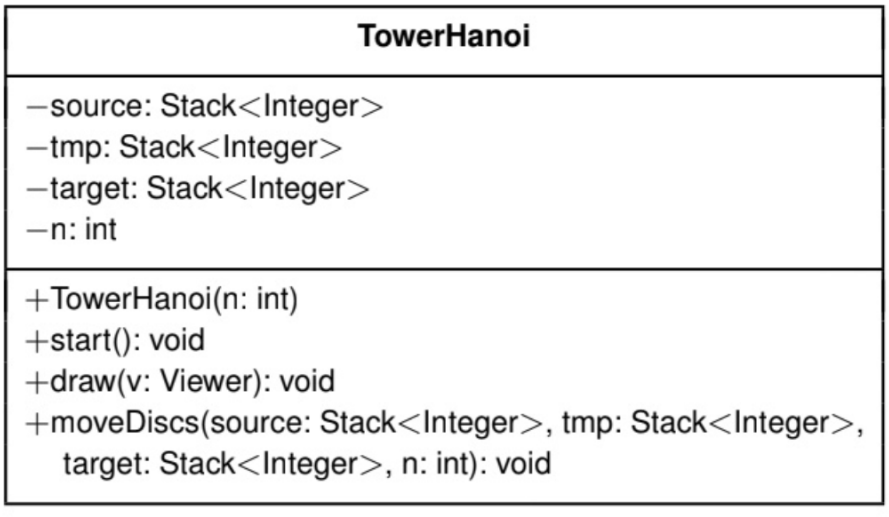

Figure 4.7: UML diagram Tower of Hanoi

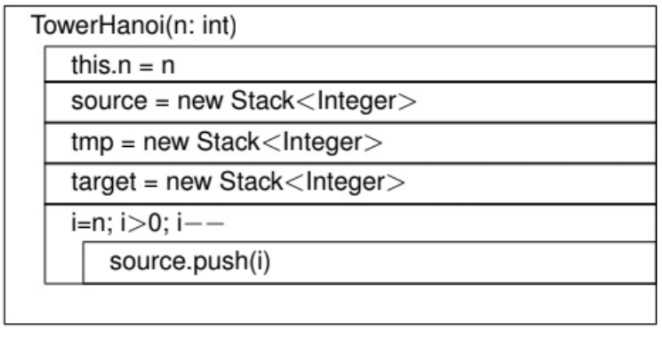

Figure 4.8: Nassi-Shneiderman diagram of constructor

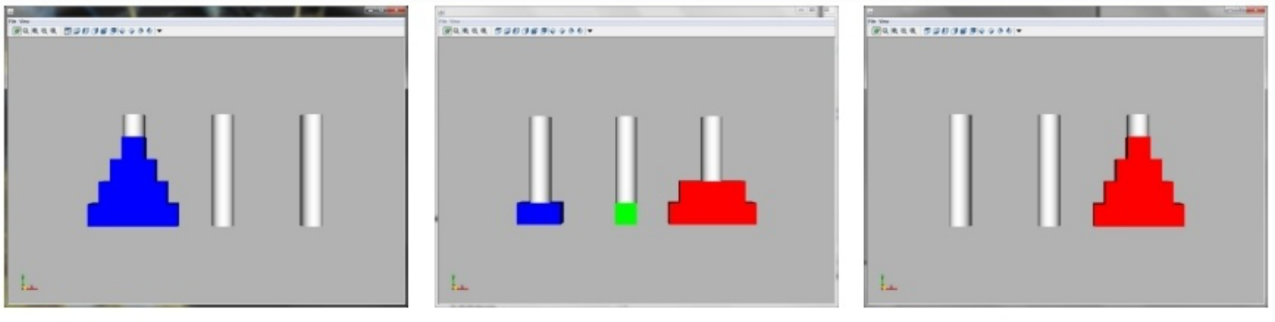

Figure 4.9: The initial state (left), final state (right) and an intermediate state (middle)

JAVA CODE

```java
package chapter1;

import java.lang.Double;
import java.util.Stack;
import inf.v3d.view.*;
import inf.v3d.obj.*;


public class TowerHanoi {

        private Stack <Integer> source;
        private Stack <Integer> temp;
        private Stack <Integer> target;
        private int n;

    public TowerHanoi (int n) {
        this.n=n;
        Stack <Integer> source = new Stack <Integer> ();
        this.source = source;
        Stack <Integer> temp = new Stack <Integer> ();
        this.temp = temp;
        Stack <Integer> target = new Stack <Integer>();
        this.target=target;
        for (int i=n; i>0; i--) {
            source.push(i);
        }
    }
```

```java
    public void start() {
        this.moveDiscs(source, temp, target , n);
        }

    public void moveDiscs (Stack <Integer> source, Stack <Integer> temp, Stack <Integer> target, int n) {

        if (n==0) {
            return;
        }

            moveDiscs(source, target, temp, n-1);
            System.out.println("Step 1");
            target.push(source.pop());

            this.print();
            System.out.print("----------------");

            Viewer view = new Viewer();
            this.draw(view);

            System.out.println("Step 2");
            moveDiscs(temp, source, target, n-1);

    }
```

```java
void print () {
    System.out.println("Source = " + source);
    System.out.print("Temp = " + temp);
    System.out.print("Target = " + target);
}
```

```java
public void draw(Viewer v) {
    Cylinder ColSource = new Cylinder(0.0, 0.0, 0.0, 0.0, 10.0, 0.0);
    Cylinder ColTemp = new Cylinder(7.0, 0.0, 0.0, 7.0, 10.0, 0.0);
    Cylinder ColTarget = new Cylinder(14.0, 0.0, 0.0, 14.0, 10.0, 0.0);
    v.addObject3D(ColTarget);
    v.addObject3D(ColTemp);
    v.addObject3D(ColSource);

    Cylinder [] Disks = new Cylinder[n];

     double R = 1;
     double H = 1;
     if (this.source.empty()==false) {
     for (int i = 0; i < source.size(); i++) {
     Disks[i] = new Cylinder (0.0, i*H, 0.0, 0.0, (i+1)*H, 0.0);
     Disks[i].setRadius(R*source.get(i));
     Disks[i].setColor("blue");
     v.addObject3D(Disks[i]);
     }
     }
```

```
  if (this.temp.empty()==false) {
  for (int i = 0; i < temp.size(); i++) {
  Disks[i] = new Cylinder (7.0, i*H, 0.0, 7.0, (i+1)*H, 0.0);
  Disks[i].setRadius(R*temp.get(i));
  Disks[i].setColor("green");
  v.addObject3D(Disks[i]);
  }
  }
  if (this.target.empty()==false) {
  for (int i = 0; i < target.size(); i++) {
  Disks[i] = new Cylinder (14.0, i*H, 0.0, 14.0, (i+1)*H, 0.0);
  Disks[i].setRadius(R*target.get(i));
  Disks[i].setColor("red");
  v.addObject3D(Disks[i]);
  }
  }

  v.setVisible(true);

 }
```

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    TowerHanoi game = new TowerHanoi(5);

    game.start();
    System.out.println("Complete");
    game.print();
    System.out.println("-------------");

    }
}
```