INTRODUCTION TO COMPUTER SCIENCE II
LAB 3
Towers of Hanoi

Time to review method calls and, while were at it, recursion. Oh, and arrays. And pointers to arrays. Yes, all of that.

1 A classic puzzle

The *Towers of Hanoi* puzzle is a classic example for the use of recursive programming. Use Wikipedia and other online resources to read about the puzzle, if it is unfamiliar; additionally, examine the recursive solutions, available in many programming languages (including Java).

Your goal, below, will be to adapt one of those recursive solutions to this puzzle into existing code.

2 Getting started

If you are doing your work on `remus/romulus`, login with Remote Desktop, open a terminal window, create a directory for your work, and change into it. Next, grab the initial source code for this assignment:

- **On `remus/romulus`:** Copy the source code with the following command...

```
$ cp ~sfkaplan/public/COSC-112/lab-3/Towers.java .
```

- **On your own computer:** Download the source code by clicking here.

Open, examine, compile, and run the code provided as needed to understand what is already there. You will see that this code does the following:

- Create three arrays of `int` that represent the towers. Each of these tower-arrays are pointed to from an array of `int[]`. That is, an array of three arrays of integers is created.
- The size of each tower (that is, the length of each array of integers) matches the number of rings the puzzle is directed to use by the user.
- Likewise, the integers themselves represent the rings, where a value of `0` indicates no ring (an unoccupied space on the tower), and a positive value indicates a ring of that size. If there are n rings in the puzzle, the rings are sized from 1 to n.
- There is a `print()` method that can print out a textual representation of the towers and the rings on them.

3 Your assignment

Notice that there is a `solve()` method that calls a `doSolve()` method. The first is a *stub method* whose job it is to make the first recursive call to the second. It is the second method (`doSolve()`) that drives the recursive solving. **Your task** is to complete the `doSolve()` method, printing the towers after each move, such that the sequence of moves that solves the puzzle is shown.

Note that you are encouraged to write as many *helper methods* as you need for `doSolve()` to do its work. Break down the task of moving the disks themselves into small methods that call one another.

4 How to submit your work

Use the CS submission systems to submit your work. Specifically, you will need to submit your `Towers.java` file. You may use either of the following two methods, while connected to `remus` or `romulus`, to use the submission system:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `cssubmit` command at your shell prompt, like so:
  ```
  $ cssubmit Towers.java
  ```

This assignment is due on Sunday, Feb-12, 11:59 pm.