Assignment 1: The Towers of Hanoi!

Out: 19th September 2016 **Due: 28th September 2016 at 11:59pm**

The Towers of Hanoi is a famous problem in computer science, begging to be visualized. In this assignment you will create a visualization of a solution to this problem.

The Problem

There are three towers A, B, C (from left to right). Tower A contains a stack of 'n' disks, kept on top of each other in descending order of radii (they have the same thickness). The problem is to transfer all 'n' disks from A to C, possibly using B as an intermediate, such that at no time is a bigger disk stacked on top of a smaller disk.

The Solution

The Towers of Hanoi is a famously recursive problem. Intuitively, a solution can be summarized as: "To transfer n disks from A to C using B, transfer the first n-1 disks from A to B using C, then transfer disk 'n' from A to C, and finally transfer n-1 disks from B to C using A." The first and last parts are recursive.

A solution in Java and C++ has been provided to you with this assignment. The solution exports the generated solution to a file. The file contains the number of moves, followed by each move (the topmost disk from the first tower to above the topmost disk of the second tower).

What to do

In this assignment you must create a program that shows the Towers of Hanoi problem being solved as an animated visualization. Your program should have the following features:

1. Each disk must be shown as a rectangle. All disks must have the same widths, but must have different lengths as per the problem. (10 points)
2. Each disk must be of a different color. (10 points)
3. The program should show the solution as an animation (60 points):
    a. It should start with all the disks being on the left tower and end with all the disks on the right tower. (10 points)
    b. The program should animate each step of the solution. A step should start when the user presses the Space bar and stop at the end of the step (for the user to press the Space bar again). (15 points)
    c. The animation should show disks flying between towers one at a time exactly as per the solution to the problem (the path a disk follows from its current to its target position is up to you). (25 points)
    d. Once the solution is complete, the user should be able to press Space bar to restart the visualization without having to run the program again. (10 points)
4. The program should take the number of disks as a command-line argument. Your program should check for invalid arguments. (10 points)

Documentation (10 points)

Submit a description of your program that has the following:

1. Screen captures of the program in action.
2. Explain how to use your program once it is run (a user manual).
3. Explain how you produced the animation. That is, how have you designed your program so that when the user presses a Space bar, an animation of the next step is shown.

Extra credit features

1. Create an “auto” mode in your program. When the user presses ‘A’, the program shows the animation of the entire solution without stopping after each step. It should loop back infinitely. Pressing ‘D’ should go back to the “one-step-at-a-time” mode. (10 points)
2. The program should allow a user to resize the window. The visualization should not appear squished or deformed when the user resizes the window (think carefully about windows and viewports) (10 points).

Hints

It will be very helpful to have a plan and a design before you write code. The actual drawing does not involve anything more than drawing rectangles and moving them on the screen. It will take more effort to plan out how you will show the animation.

One way to think about it is this: ultimately your program must know where to draw the disks at every frame. You can think of each step taking up ‘x’ number of frames. If you precompute the positions of all these disks for each frame within a step, the actual program will simply go through them! That is, you can think about calculating the position of each disk during every frame of each step before you even start drawing.

This is called “offline processing” because when the program is animating, it is not processing any data. This isn’t always feasible depending on the data and animation, but when it is, is the best option because your program is free to concentrate only on drawing when the animation is playing. Result: fast animation!

What to submit

Submit the entire IntelliJ/Qt Project folder in a single zipped file. Set up the program so that we should be able to run it as-is and see an animation. Include a README file that lists which features you were able to complete, and which ones you could not.

You **do not** need to submit the Towers of Hanoi solution project given to you.