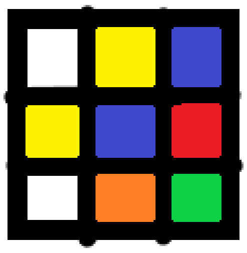
Running time analysis:

The Listz, Entry, and CubeDisplay class have to process the most information.
The Listz class's worst case running time would be O(n). This is because in order to add/remove an entry, the class would have to traverse and get the next element in order to find the index.
The Entry class is responsible for sorting 2 arrays, of maximum length 12. In this case, insertion sort was used (just because of its simplicity and short time for small batches — as tested in a previous assignment). However, if this insertion sort was scaled up to sort 100 values, it would take a longer time to sort as it's time complexity is O(n^2).
The CubeDisplay class is responsible for resetting itself and then applying scrambles to the 6 2D arrays to then create a representation of the scrambled cube after following the randomly generated scramble. The act of scrambling the cube will only take O(n) time, as it takes in the size of the String array passed into the scrambling method, and processes it one element at a time. However, in order to reset each 2D array, the "faces" have to be reset back to the original "colours" (ie. blue = "b" for one entire face). This will take O(n^2) time.
Overall, the program will take O(n^2) time.

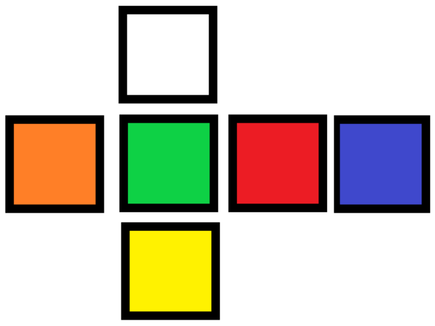
Major functions analysis:

String[] Scrambler.generateScramble()
This method generates an array of strings. It is based on the possible moves with a 2x2-5x5, consisting of Up/Down/Left/Right/Front/Back faces, ClockWise/CounterClockWise/180 Degree turns, and single layer/wide layer (2 layers) turns. This gives a combined total of 36 possible moves to generate. The scramble length is based on average scramble lengths used in World Cube Association (WCA) competitions. Upon being called, the function will pick at random, from 3 different groups: face, direction, and number of layers to turn (as mentioned above). This will repeat for the predetermined scramble length. The array of strings generated will be used by the CubeDisplay class.

CubeDisplay



The cube is represented by 6 2D arrays (ie.). The cube is oriented with the lightest face (usually white) on the top, and the darkest face (usually green) on the front, before scrambling, as per WCA regulations.


(most cubes will follow this colour scheme shown here)

Scrambling the cube is actually broken down into 2 sections: rotating a face, and swapping layers that are affected by the turning of a face.
But before the cube is scrambled, it must first be reset.

Void CubeDisplay.reset()
Upon being called, this creates new 2D arrays of the size (n*n) based on the cube's current size. Then, each element in each face is given its solved state's colour (ie. if it's the green face, then all elements in the 2D array are 'g'). This method also creates new temporary variables of the same size as the cube in order for those to be used later on (storing and swapping values).

Void CubeDisplay.scramble(String[] scram)
Once this method is called, it will read from the given array of strings, one by one. It'll check for what kind of move it is (ie. Up face, clockwise, wide move), and apply a swap of the layers that are affected. It will also call the respective method to move the face in the correct direction (ie. CubeDisplay.cwFace())

Char[][] CubeDisplay.cwFace/ccwFace/cw2Face(char[][] face)
Once this method is called, it will return a char value based on the input. It will take the given face, and rotate the faces accordingly (ie. when the cube is being rotated CW, it will move the top row of the face to the right column, the right column to the bottom row, the bottom row to the left column, and the left column to the top row). It will then return the new modified face.

Notes:
Averages of 5 and averages of 12 are calculated by removing the best and worst time, and taking the mean of the remaining times.

The AnimationTimer is taken from James_D from stack overflow, found here:
https://stackoverflow.com/questions/34801227/fast-counting-timer-in-javafx
This block of code finds the time elapsed and displays it to a label, constantly updating it's value, essentially becoming a stopwatch.