```java
/**
 * This program simulates a Rubik's cube competition format, one
 * of my favorite pass-times, in a casual setting. After running the program,
 * it gives a user a randomized scramble for their Rubik's cube, similar to a real
 * competition. It utilizes a random sequence of symbols that signify the orientation of the side that
 * should be turned. After giving the user a random scramble,the program gives the user 15 seconds
 * of examination time. After the 15 seconds are over, the user is able to press the 'enter' key
 * and start the timer. To stop the timer, the user must press the 'enter' key again, where they are
 * given the option to continue solving Rubik's cubes by pressing 'Y' or the option to stop the session
 * by pressing 'N'. After the session is ended the user, they are given an evaluation of their times during the
 * session, including a list of all their times, their best time, and an average of their times.
 *
 *
 * @ Alexander Yuan
 * @ 5/20/2020
 */

import java.util.Scanner;
import java.io.File;
import java.io.IOException;
import java.io.*;
import java.util.*;
import java.lang.Math;
import java.io.DataInputStream;
import java.util.concurrent.TimeUnit;

public class CubeTimer
{
    public static void main (String [] args) throws IOException
    {
        System.out.println("Welcome to my Rubik's Cube Timer!");
        System.out.println();
        Scanner in = new Scanner(System.in);
        boolean tryAgain = true;
        String answer = "";
        float cubeTime = 0;
        ArrayList <Float> listOfTimes = new ArrayList<Float>();
        //calls the GetScramble method to randomly generate a Rubik's cube scramble for the user
        GetScramble();
        // calls the examineTime method to give the user 15 seconds of examination time for their scramble
        examineTime();
        // while tryAgain variable remains true, it continues to loop through this function,
        // allowing the user to solve the cube again and hopefully achieve a better and faster time
        while (tryAgain)
        {
            // calls the cubeTimer method to time the user
            cubeTime = cubeTimer();
            // adds the time to a list that will later be utilized with various methods as a parameter
            listOfTimes.add(cubeTime);
            System.out.println();
            System.out.println("Would you like to try again? Please enter Y or N.");
            answer = in.nextLine();
            // if the user presses 'N', the session will end and conditions to the while loop
            // will no longer remain satisfied. If the user presses 'Y', the session will continue
            // as the conditions to the while-loop remain satisfied and the user will be able to
            // time another solve, hopefully achieving a faster time.
            if (answer.equalsIgnoreCase("N"))
            {
                tryAgain = false;
            }
        }
        // calls the GetBestTime method utilizing a list of times in the session, and returns the fastest time
        float bestTime = GetBestTime(listOfTimes);
        //calls the GetAverage method utilizing a list of times from the session, and returns the average time in the session
        float averageTime = GetAverage(listOfTimes);
        //a final evaluation and summary of the times achieved during the session
        System.out.println("Congratulations!!!");
        System.out.println("Your list of times from this session: " +listOfTimes);
        System.out.println("Your best cubing time is " + bestTime + " seconds.");
        System.out.println("Your average cubing time is " + averageTime + " seconds.");
    }

    // a method that returns the best time in the list
    public static float GetBestTime(ArrayList <Float> a)
    {
        float bestTime = 0;
        for (int i = 0; i < a.size(); i++)
        {
            // reads the time from the list; if the time is lower of faster than the current bestTime,set bestTime as that time
            // if bestTime does not have a value assigned to it, set the current time as bestTime
            if (a.get(i) < bestTime || bestTime == 0)
            {
                bestTime = a.get(i);
            }
        }
        return bestTime;
    }

    // a method that returns the average time in the list
    public static float GetAverage(ArrayList <Float> a)
    {
        float totalTime = 0;
        float averageTime = 0;
        // utilizes a for-loop to run through each item in the list and add it to a total
        for (int i = 0; i < a.size(); i++)
        {
            totalTime += a.get(i);
        }
        // calculates average with average = total / num of items
        averageTime = totalTime / a.size();
        return averageTime;
    }

    // a method that gives the user a randomized scramble.
    // It utilizes a random sequence of symbols that signify the side or face that should be turned
    // 'R' indicates right side, 'L'indicates left side, 'U' indicates top, 'D' indicates bottom
    // 'F' indicates front, 'B' indicates back
    // In addition, sometimes a symbol is followed by a ' or a '2' to indicate how many times the face should be turned and in what
    // orientation.
    public static void GetScramble()
    {
        Scanner in = new Scanner(System.in);
        System.out.println("Please press 'Enter' for a randomized scramble!");
        in.nextLine();

        // array for how many times a face should be turned and in what orientation so that it can be
        // randomly selected, independent from which face should be turned
        String [] twistAmount = new String[3];
        twistAmount[0] = "";
        twistAmount[1] = "'";
        twistAmount[2] = "2";

        // array for which face should be turned so that it can be randomly selected, independent
        // from how many times a face should be turned and in what orientation
        String [] moveType = new String[6];
        moveType[0] = "R";
        moveType[1] = "L";
        moveType[2] = "U";
        moveType[3] = "D";
        moveType[4] = "F";
        moveType[5] = "B";
        String scramble = "";
    // for-loop to randomly select symbols indicating face and orientation to be turned
    // adds these random symbols to a string of 12
    for (int i = 0; i < 12; i++)
    {
        int randomTwist = (int)(Math.random() * 3);
        int randomMove = (int)(Math.random() * 6);
        scramble = scramble +moveType[randomMove] +twistAmount[randomTwist] +" ";
    }
    System.out.println(scramble);
}

// a method that gives the user 15 seconds of examination time before solving the cube and allows them to
// interact with the program
public static void examineTime()
{
    Scanner in = new Scanner(System.in);
    System.out.println();
    System.out.println("Please press 'Enter' to start examination time.");
    in.nextLine();

    // retrieves the current system time in milliseconds as a starting point
    long startTime = System.currentTimeMillis();

    long currentTime;
    long secondsPassed = 0;

    // start counting down from 15 seconds
    System.out.print("15...");

    // keep counting down from 15 seconds using the system time
    while(secondsPassed < 15)
    {
        try
        {
            // wait for 1 second
            TimeUnit.SECONDS.sleep(1);
        }
        catch (InterruptedException e)
        {
        }

        //get the current system time in milliseconds
        currentTime = System.currentTimeMillis();

        // calculate elapsed time from the starting point
        long eTime = currentTime - startTime;

        //convert milliseconds to seconds
        secondsPassed = eTime / 1000;

        // output the  countdown
        System.out.print((15 - secondsPassed) +"...");

        // when the countdown reaches 0, print "Time's up!"
        if ((15 - secondsPassed) == 0)
        {
            System.out.println("Time's up!");
        }
    }
}

public static float cubeTimer() throws IOException
{
    Scanner in = new Scanner(System.in);
    System.out.println();
```

```
        System.out.println("Please press 'Enter' to start timing!");
        System.out.println("To stop timing, press 'Enter' again!");

        in.nextLine();

        long startTime = System.currentTimeMillis();
        long currentTime;
        long elapsedTime = 0;
        long etimeInSeconds = 0;
        long etimeInSeconds1 = 0;

        DataInputStream dis = new DataInputStream(System.in);

        while(etimeInSeconds < 300)
        {
            // timer is less than 5 minutes

            // check if user pressed the 'enter' key
            if (dis.available() != 0)
            {
                // user pressed the 'enter' key to end the timer
                String strInput = dis.readLine();

                // print the final time
                System.out.println("Your final time: " +
                (float)elapsedTime / 1000 + " seconds.");
                break;
            }

            // get system current time to calculate the elapsed time from the starting point
            currentTime = System.currentTimeMillis();
            elapsedTime = currentTime - startTime;

            // convert elapsed time into seconds
            etimeInSeconds = elapsedTime / (1000);
            if (etimeInSeconds >= etimeInSeconds1 + 1) {
                // print the elapsed time in seconds when the elapsed increases by 1 second
                System.out.println(etimeInSeconds);
                etimeInSeconds1 = etimeInSeconds;
            }
        }
        return (float)elapsedTime / 1000;
    }
}
```