HW 2: Transform Calculator

In this assignment, you will make a program to transform points using matrices. It is assumed that you will use the Java OpenGL Matrix Library (JOML). The main class you create should be called TransformCalculator, though you may also create helper classes if you wish.

You will need to ensure that your classpath includes the JAR file joml-X.XX.X.jar, in which the Xs represent the version. This will allow you to import classes from the package org.joml. You will find especially useful those that implement vertices and matrices. The full documentation for JOML is located at https://javadoc.io/doc/org.joml/joml/latest/. (Also note that this package contains a class called Math. If you’re not careful, this may conflict with java.lang.Math.)

Your program will invite the user to enter a series of transforms, which are all strings of 1-2 letters followed by several numeric parameters (indicated with <>s). The program will be case-insensitive, and elements of the transforms will be separated by any amount of whitespace. This will continue until the user enters a blank line. Here are the possible transforms:

- **T <Δx> <Δy> <Δz>:** Translate the point. The three parameters indicate the distance to translate in each of the three dimensions.
- **S <σ>:** Scale relative to the origin, by a factor of σ.
- **S <σx> <σy> <σz>:** Scale relative to the origin, but by different factors in the three dimensions.
- **RX <θ>:** Rotate by <θ> degrees, around the x-axis.
- **RY <θ>:** Rotate by <θ> degrees, around the y-axis.
- **RZ <θ>:** Rotate by <θ> degrees, around the z-axis.
- **L <cx> <cy> <cz> <tx> <ty> <tz>:** Rotate and translate to camera space with a “look-at” transform, such that the camera is at (0, 0, 0) and looking in the −z direction. <cx>, <cy>, and <cz> are the camera’s original position, and <tx>, <ty>, and <tz> are the position of the target, that the camera is looking at.
- **O <w> <h> <dn> <df>:** Scale and translate to perform an orthogonal projection. All visible points will be within the box between (−1, −1, −1) and (1, 1, 1). <w> is the width of the viewing solid, <h> is its height, <dn> is the distance from the camera to the near clipping plane, and <df> is the distance to the far clipping plane.
- **P <φ> <a> <dn> <df>:** Perform the non-affine perspective transform. All visible points will be within the box between (−1, −1, −1) and (1, 1, 1). <φ> is the field of view in the horizontal direction, <a> is the aspect ratio, <dn> is the distance from the camera to the near clipping plane, and <df> is the distance to the far clipping plane.
- Any unidentified or otherwise illegal transform (e.g. the wrong number of parameters) should result in a friendly error message on stderr, and be ignored.
After the user has entered a blank line, the program will output the $4 \times 4$ transform matrix that has been calculated. It will then request triples, one line at a time with individual values separated by any amount of whitespace. Each triple will be treated as a point and transformed by the matrix. Illegal requests or points without 3 values will elicit a friendly error message on stderr, and should not be transformed. Transforms should be clearly marked, and should be offset by a tab. It will continue until the user enters a blank line. When this happens, the program will terminate.

Here is how the program might appear:

```
Please enter the desired transforms, and enter a blank line when done:
T -1 -2 -3
RY 5
T 1 2 3

The final transform matrix is:
\[
\begin{bmatrix}
0.996 & 0.087 & 0 & -2.577E-1 \\
0.000E+0 & 1.000E+0 & 0.000E+0 & 0.000E+0 \\
-8.716E-2 & 0.000E+0 & 9.962E-1 & 9.857E-2 \\
0.000E+0 & 0.000E+0 & 0.000E+0 & 1.000E+0 \\
\end{bmatrix}
\]

Please enter 3D points to transform, and hit enter when done.
0 0 0
TRANSFORMED: 0.25766194 0.0 -0.098571695
5 6 7
TRANSFORMED: 5.848726 6.0 6.4390125
4 q 4
I’m sorry. "4 q 4" is not a valid point.

Goodbye!
```