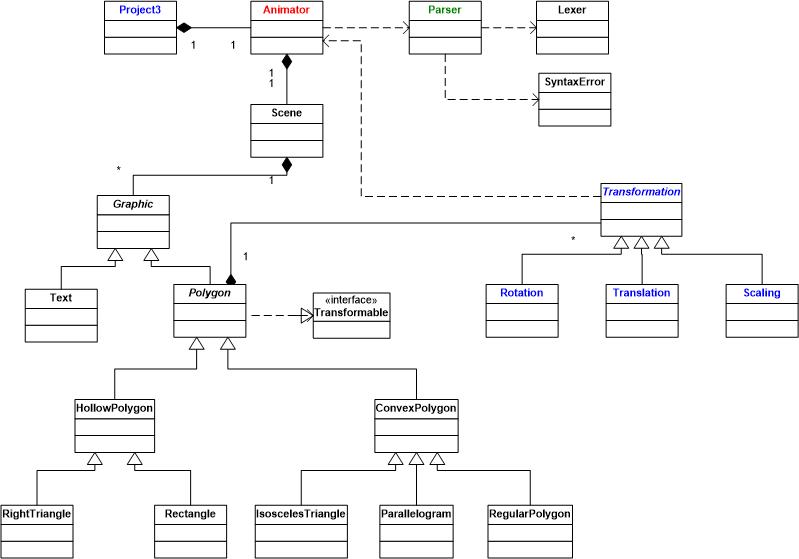
**Specification**  
The third project builds on the second project by adding the ability to specify animations of the transformations. The UML diagram for the new version of the project is shown below:



All the classes shown in black are unchanged from project 2.

The classes shown in blue will require the following modifications:

1. Two new callback functions must be added to the main project3.cpp source file. The first function is one that is called when a timer is fired. It should call thenextStep function of the Animator class discussed below and set the timer to refire. The second callback function is one that is called when a character is typed on the keyboard. It should call the keyboard function of the Animator class passing the key that was typed. The draw callback function must be modified to call the draw function in the Animator class. The Scene object previously declared in this file must be replaced by an Animator object.
2. The Transformation interface from project 2 must now become an abstract class. It will require variables that define the starting and stopping steps of the animation and a constructor to initialize these variables.
3. The Rotation class must be modified so the the transform function computes the rotation angle by multiplying the angle of rotation by the step of the animation, before performing the rotation.
4. The Translation class must must be modified so the the transform function computes the translation distances by multiplying the translation distances by the step of the animation, before performing the translation.
5. The Scaling class must must be modified so the the transform function computes the scale factors by raising the scale factor to the power defined by the current step of the animation.

There is only one new class shown in red, Animator. It should have the following data elements.

1. The Scene object that was previously declared in project2.cpp.
2. A Boolean variable that indicates whether or not an animation is currently in progress.
3. A counter that keeps tracks what step of the animation is currently being drawn for active animations.

It should have the following functions:

1. A constructor that is passed the name of the scene file that parses the scene and initially creates the window.
2. A function draw that draws the scene. Because double buffering should now be used, it should swap the buffers after performing the drawing. It should be called by the draw callback function in project3.cpp.
3. A function named nextStep that advances to the next step of the animation. If an animation is active, it should advance to the animation step and redraw the scene. It should be called by the timer callback function in project3.cpp.
4. A function named keyboard that controls starting, pausing, and resetting the animation. It should start the animation when the letter *s* is typed, pause it when *p*is typed and reset the animation back to the initial state when an *r* is typed. This function needs to be called by the keyboard callback function inproject3.cpp.

The class shown in green, the Parser class, is provided in the attached .zip file. The new version of this class has been modified to add the option of including animation steps on each transformation. The revised grammar is shown below, with the changes highlighted in red:

scene -> SCENE IDENTIFIER number\_list graphics END '.'  
 graphics -> graphic graphics | graphic  
 graphic -> text | transformable\_graphic transformations END  
 transformable\_graphic -> right\_triangle | rectangle | isosceles | parallelogram | regular\_polygon  
 right\_triangle -> RIGHT\_TRIANGLE COLOR number\_list ';'  
 rectangle -> RIGHT\_TRIANGLE COLOR number\_list ';'   
 isosceles -> ISOSCELES COLOR number\_list ANGLE NUMBER ';'  
 parallelogram -> PARALLELOGRAM COLOR number\_list ANGLE NUMBER ';'  
 regular\_polygon -> REGULAR\_POLYGON COLOR number\_list SIDES NUMBER RADIUS NUMBER ';'   
 text -> TEXT COLOR number\_list AT number\_list STRING ';'  
 transformations -> transformation transformations | transformation  
 transformation -> action steps  
 action -> rotation | scaling | translation  
 steps -> STEP NUMBER TO NUMBER ';' | ';'  
 rotation -> ROTATE ANGLE NUMBER ';'  
 scaling -> SCALE number\_list ';'  
 translation -> TRANSLATE number\_list ';'  
 number\_list -> '(' numbers ')'  
 numbers -> NUMBER | NUMBER ',' numbers

The new Token.h file that includes the STEP and TO tokens is also included in that .zip file. **Sample Input and Output**  
Below is a sample of a scene definition file that would provide input to the program:

Scene Polygons (500, 500)  
 Parallelogram Color(0.0, 1.0, 0.0) Angle 80;  
 Scale (20.0, 20.0);  
 Translate (-0.2, 0.2) Step 1 to 25;  
 Scale (1.3, 1.3) Step 25 to 28;  
 Rotate Angle 30 Step 31 to 42;  
 End  
 Isosceles Color (0.0, 0.0, 1.0) Angle 90;  
 Scale (1.5, 1.5) Step 61 to 71;  
 Translate (0.1, 0.1) Step 72 to 80;  
 Rotate Angle 45 Step 81 to 84;  
 End  
 RegularPolygon Color(1.0, 0.0, 0.0) Sides 4;  
 Scale (50.0, 50.0);  
 Rotate Angle 36.0 Step 91 to 100;  
 End  
 Text Color(0.0, 0.0, 0.0) at (5.0, 150.) "Hello World";  
 End.