

**ADVDISC**

Machine Project 1 Documentation

|  |  |
| --- | --- |
| **Section** | S19 |
| **Team Members** | Amadora, Angelo John  Andres, John Joseph  Fernandez, Ryan Austin  Syfu, Jonah Espiritu |
| **Date Submitted** | October 26, 2015 |

**Table of Contents**

I. Contribution of Each Member / 3

II. Introduction / 5

III. Design / 6

IV. Implementation / 13

V. Conclusion / 14

VI. References / 15

Appendix A: Objects and their Transformations / 16

**I. Contribution of Each Member**

|  |  |  |
| --- | --- | --- |
| **Member** | **Code** | **Other** |
| Andres, John Joseph | Ellipse  Point | Quality Assurance and Testing |
| Amadora, Angelo John | Parabola  Hyperbola | Documentation |
| Fernandez, Ryan Austin | Graph  AbstractCreatePanel  CreateEllipsePanel  CreateHyperbolaPanel  CreateLineSegmentPanel  CreateParabolaPanel  CreatePointPanel  CreatePolygonPanel  GraphPanel  MainGraphicsPanel  TransformPanel  TransParamPanel  ReflectPanel  RotateSetPanel  ScalePanel  ShearRotatePanel  TranslatePanel  GraphicsFrame  DrawEllipse  DrawHyperbola  DrawLineSegment  DrawParabola  DrawPoint  DrawPolygon  DrawVector  Driver  Ellipse  GraphicsController  AGBLayout  Hyperbola  Parabola  Reflect  RotateDouble  RotateSet  Scale  Shear  TransformPanelBuilder  TransformPanelDirector  Translate  TransParamPanel.InputListener DoubleRotateObject2D  IController  IDraw  ITransform  Object2D  AdvancedObject2D  ShearObject2D  ShearObject2D  TransformBuilder  Shape  Transformation  Graph2  CreateMenu  draw.GraphPanel2  TransformPanel2  CreateFrame  CreatePanelFactory  GraphicsController2  SelectionPanel  TransformPanelBuilder2  Curve  DrawCurve | Research  Documentation |
| Syfu, Jonah Espiritu | LineSegment  Polygon  Vector | Quality Assurance and Testing |

**II. Introduction**

Linear Algebra is a very useful field of mathematics, especially in the field of computer science, particularly the concept of linear transformations. Linear transformations are used liberally in the field of computer graphics.

In computer graphics, multiple objects can be drawn on the screen, which includes, but is not limited to points, vectors, line segments, polygons, and conic sections.

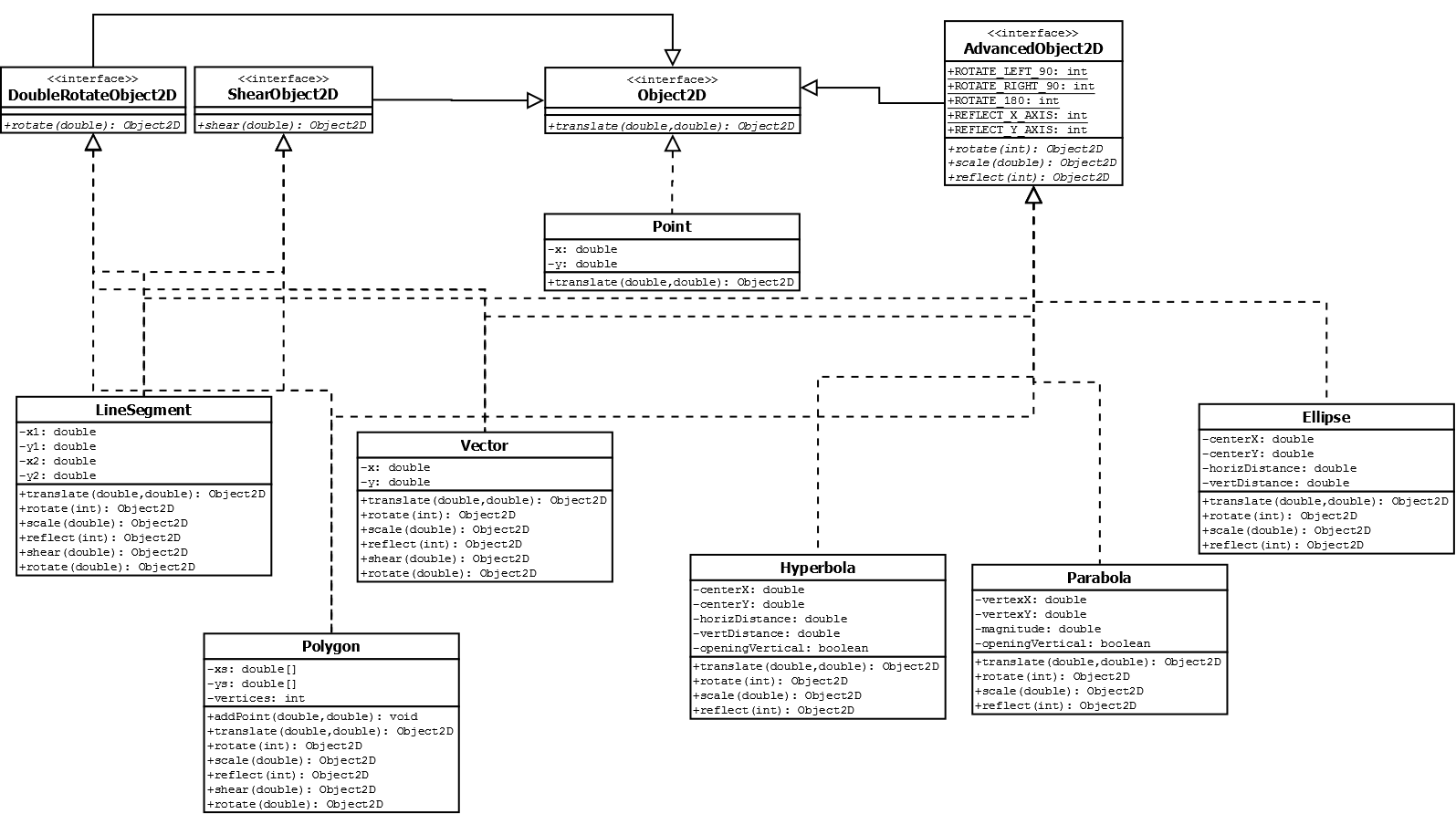
Specific transformations can be performed on these objects such as shearing, translating, rotating, scaling, and reflecting. By representing the original objects as an operation performed on matrices, and representing these transformations as transformation matrices, you can perform these operations on the objects.

This project aims to demonstrate the performance of these transformations on the given objects using the Java programming language.

**III. Design**

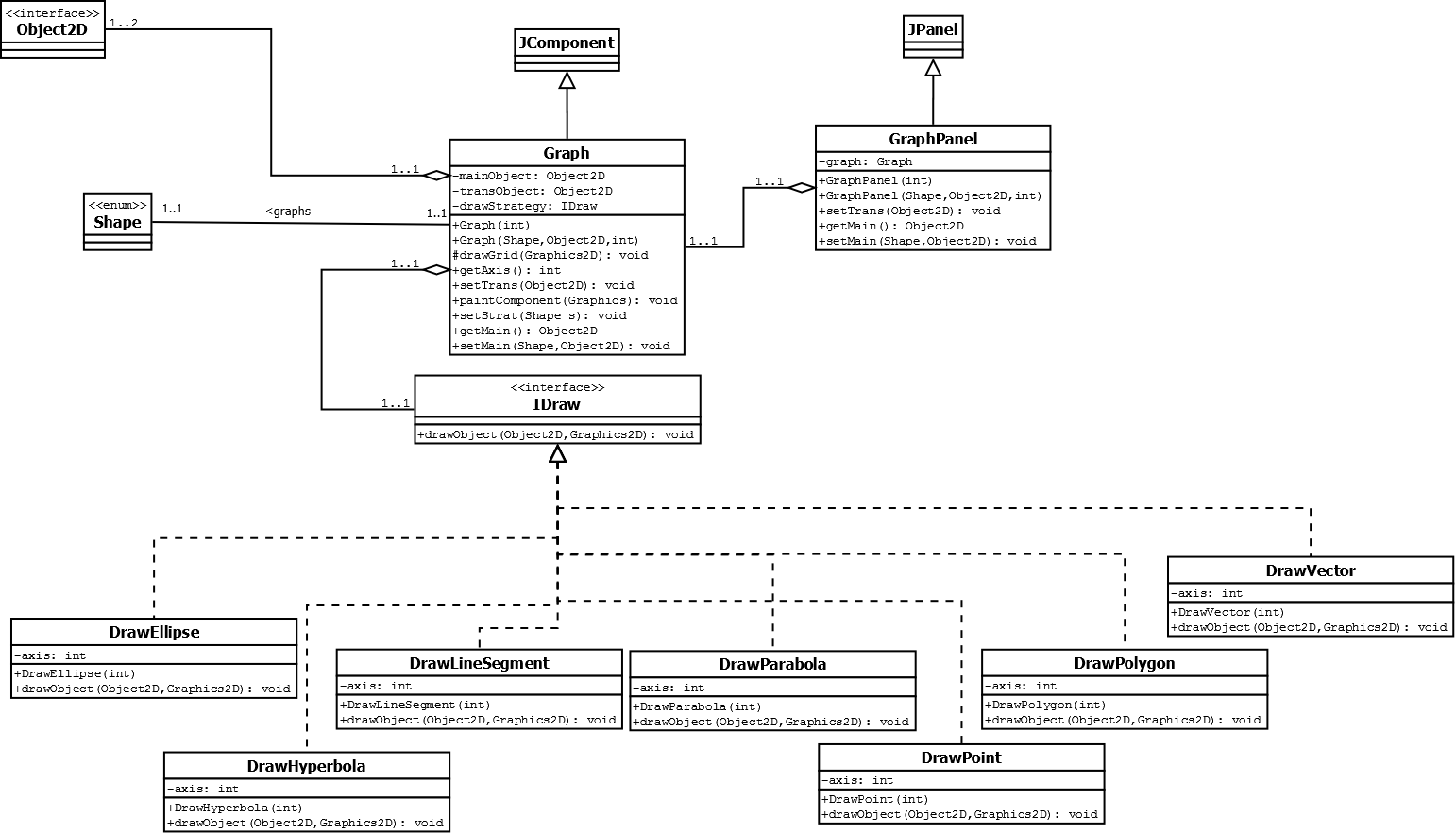
**Software Architecture**

The software follows the Model-View-Controller architecture. The software has five main modules. These modules are the object model module, under model; the drawing module, the creation module, and the transformation module, all under view; and the controller module.



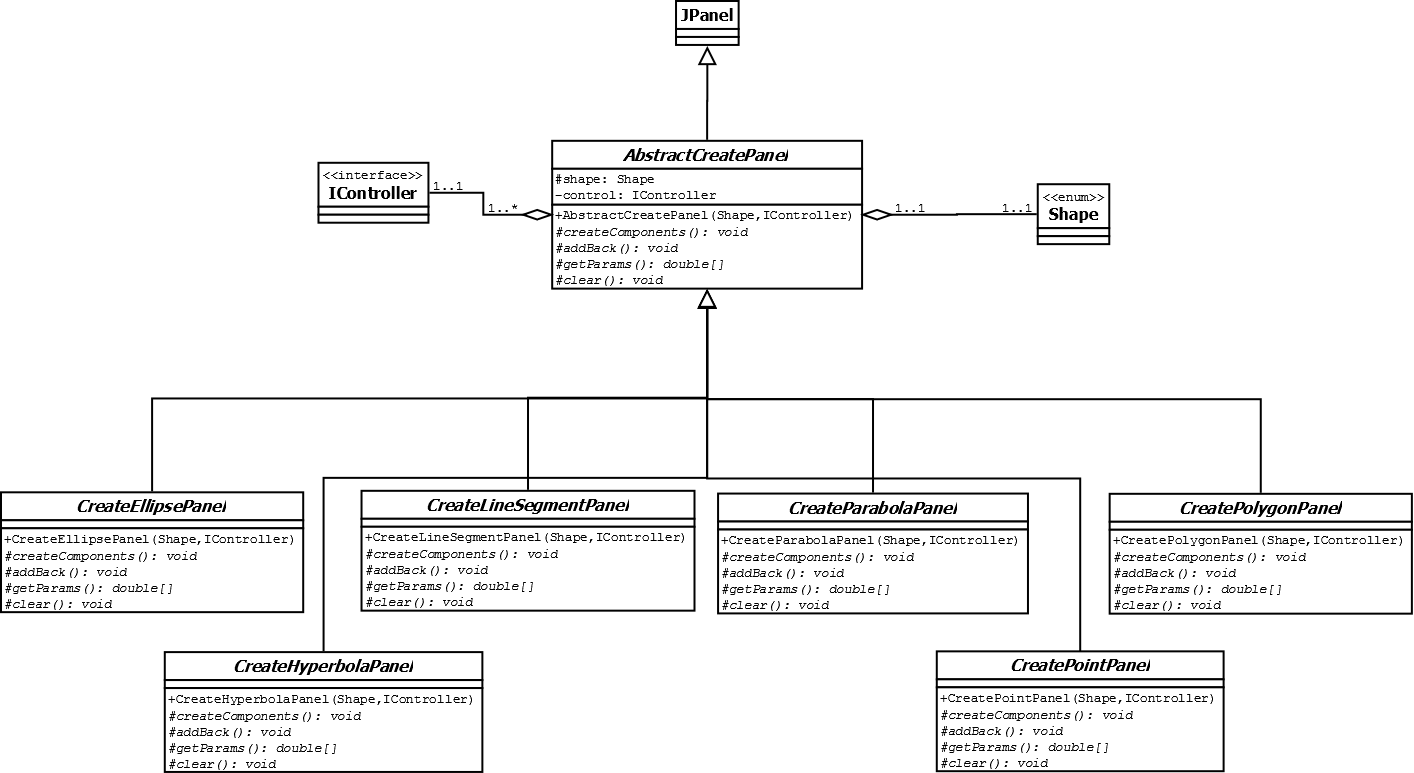
**Figure III-1. – Class Diagram for the Object Model Module**

The object model module has one main interface called the Object2D which allows for translation. It has three subinterfaces, DoubleRotateObject2D, ShearObject2D, and AdvancedObject2D. This is because it would be difficult to draw the sheared or liberally rotated conic sections so, following Interface Segregation Principle, these operations were separated into their own interfaces. AdvancedObject2D only allows rotation by multiples of 90 degrees and reflection by x-axis and y-axis. There are seven classes representing the seven kinds of objects. Point only implements Object2D, the conics only implement AdvancedObject2D, and LineSegment, Polygon, and Vector implement all the subinterfaces.

****

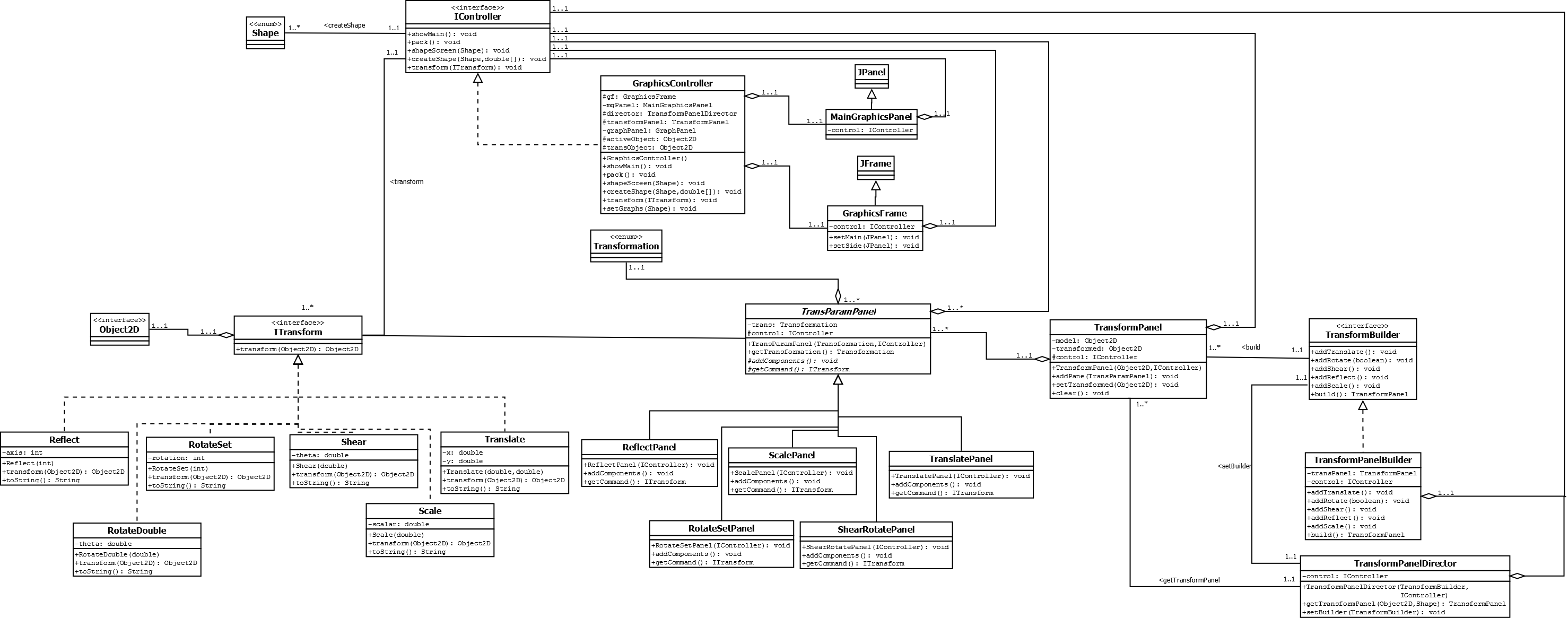
**Figure III-2. Class Diagram for the Draw Module**

The Draw Module consists of a Graph object, which handles all drawing, contained in a GraphPanel, which is added to the main JFrame. The Graph draws using a certain strategy, implemented as the Strategy Design Pattern. Each kind of shape has an implementation under the strategy interface IDraw.



**Figure III-3. – Class Diagram for the Creational Module**

The creational module follows the Template Method Pattern with the base class AbstractCreatePanel. The constructor and the ActionListener contain the final methods of the base class, involving placing components and getting the parameters inputted by the user. Each panel then verifies their own input and returns an array of parameters inputted that will be interpreted by the controller.



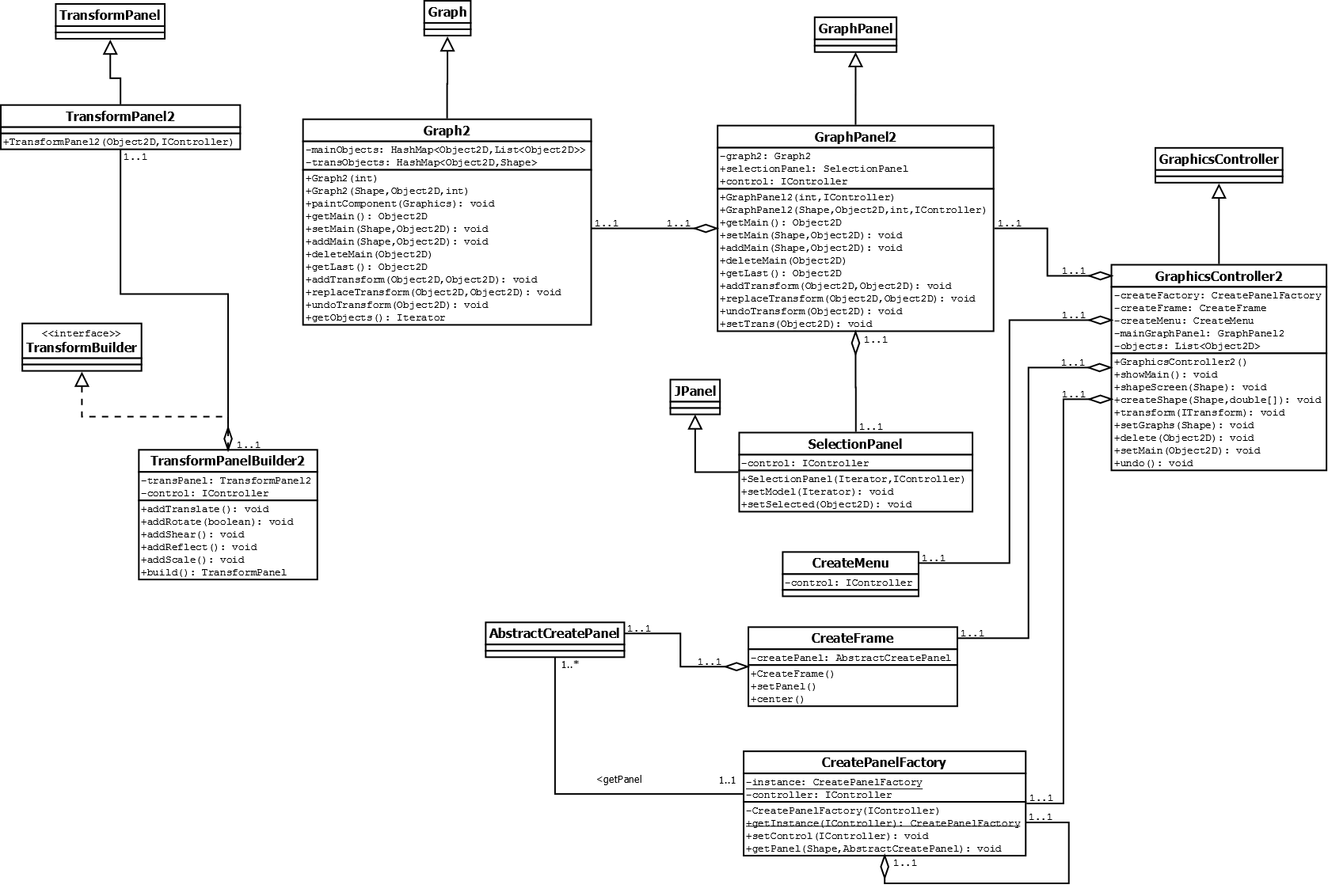
**Figure III-4. – Class Diagram for the Transformation and Control Module**

Transformations are represented using Command Objects, following the Command Design Pattern. Each Command object takes an Object2D and returns the transformed object. Each transformation is also represented by a TransParamPanel which allows the user to input the transformation parameters. Since each kind of object has a different set of transformations available to it, the Builder Pattern was used to build the appropriate TransformPanel with only the TransParamPanels the object needs.

The Controller module handles all connections from models to interface. A standard flow would be that the user selects a shape to add; the controller sets the view to the appropriate AbstractCreatePanel. The user inputs and creates the object; the controller creates the object and sets the draw panel as active view, drawing the object. The user selects a transformation; the controller receives the generated Command Object and transforms the activeObject, passing it to the GraphPanel, drawing it on the screen. This interaction is repeated.

**Mark II**

After careful consideration and comparisons with other groups, the team decided to add features to the machine project. The features are being able to add multiple shapes at once and being able to perform stacked transformations, as well as being able to undo transformations.



**Figure III-5. – Class Diagram for Additions in Mark II**

The main additions here are the HashMaps in Graph2, the Selection Panel, the new Creation System, the new Transform Panel, and the new Controller.

The system keeps track of multiple objects, which are mapped to Lists of objects using a Hash Map. These lists are implemented as Stacks. For each transformation, a push operation is performed on the stack. For each undo, a pop operation is performed. GraphPanel2 then retrieves the objects to update the new SelectionPanel, which allows for selection of objects. This retrieval makes use of the Iterator Pattern to abstract the data structure used by GraphPanel2.

The Selection Panel allows for selecting which object to manipulate right now amongst all of the drawn objects. It also allows for deletion of objects.

The new creation system makes use of a JMenuBar to select which kind of object to add. It then calls the factory method of CreatePanelFactory, obviously using the Factory Pattern. The CreatePanelFactory is also a singleton. The returned AbstractCreateFrame of CreatePanelFactory is then shown via CreatePanel.

The last addition is the new TransformPanel2, which simply replaces the back to Main Menu button with an undo button, which then defers the undo command to the controller, who then updates the model of stored states.

With regards to the new controller, GraphicsController2 has new methods which allow interfacing with the new GraphPanel2.

Overall, Mark II is much more flexible than Mark I given its ability to draw multiple shapes, select from these shapes, and to undo stacked transformations.

**Matrix Representation**

**Points, Vectors, Line Segments, Polygons**

Points and vectors are represented using 2x1 matrices.

For line segments, each point is defined as a 2x1 matrix, to be called v1 and v2, and the entire line segment is defined as the augmented matrix

which results in

For polygons with n sides, each vertex represented by v1,v2,…,vn, the matrix is simply all the vectors as columns.

**Conic Sections**

All conic sections are represented in the general form of a second degree equation in two variables ax2 + 2bxy + cy2 + ex + fy + h = 0. In matrix form, this is.

Or XTCX + GX = -h

Since user input is vertex or center and magnitude (for parabola) or vertical and horizontal distances from center, the standard form of the conic sections is first converted to the above form.

**Operations**

Translation is simple addition of the translation vector

to the points in question. For line segment, the matrix of v as the column twice is added. For polygons of n vertices, the matrix of v as the column n times is added. For conic sections, the equation is transformed back to standard form and the translation vector is added to the center or vertex (h,k).

Scaling is done by scalar multiplication of the scalar r to the matrices in question. For conics, they are transformed back to standard form and the scalar is multiplied to the magnitude or distances from center.

The following operations are now represented by a transformation matrix P. For points, vectors, line segments, and polygons, the new vector is Pv for the points/vectors and PA for the line segments and polygons. For conic sections, the new equation is formed by

XTP-1CPX + GPX = -h

The resultant equations are converted to the standard form and inputted directly into the program. For a complete list of resultant matrices and equations and their conversions to standard form, see Appendix A.

For rotation,

for rotation by counterclockwise.

For shearing,

for rotation by clockwise.

For reflection, given a vector to use as a reflection axis passing through the origin

The transformation matrix is

So for reflection along the x-axis, the vector is

And the matrix is

And for reflection along the y-axis, the vector is

And the matrix is

**Transforming Conics**

The only reason the other transformations were not performed was due to the drawing requirement. Once parabola and hyperbolae were successfully drawn, as will be discussed in Chapter IV, the team decided that transforming conics was feasible.

Two new classes were introduced: Curve and DrawCurve, the latter of which draws irregular curves or second degree equations in two variables where the coefficient for the term xy is nonzero. This occurs when matrix C in the conic representation is not a diagonal matrix.

Three cases were considered: when the curve is a twisted parabola, when the curve is a twisted hyperbola, when the curve is a twisted ellipse. All of these were implemented in the code via the algorithm mentioned below.

As for the operations themselves, the above transformation was explicitly implemented in code, introducing two dimensional arrays in the Curve class only. With regards to a rationale as to why this wasn’t implemented in the other object classes, it is to save time in looping through the matrices, since the operations were performed beforehand as seen in Appendix A.

**IV. Implementation**

The Java programming language was used, where Java Swing was used for the Graphical User interface. For rotation and shearing, the Math library was used to convert input to radians (toRadians(double) : double) and the sin, cos, and tan(sin(double):double, cos(double):double, tan(double):double)) functions were also used to compute for the transformation matrix.

For drawing, Graphics2D was used. Line segments were drawn using Line2D.Double. Points were drawn using the same class, where the start and end were the same point. Vectors were drawn using the same class where the origin was one of the endpoints. Polygons were drawn using the same class where each vertex was connected to the next one in the array. The Ellipse2D.Double class was used for drawing ellipses. For parabolae and hyperbolae, a method was implemented wherein given a y value for opening vertical and an x-value for opening horizontal, the method would return the two intersection coordinates for that value. The following pseudocode was then used to draw the curve

roots = get roots at boundary

i = boundary moved towards origin

While i in correct direction of vertex

nextRoots = get roots at i

draw line segments from roots to nextRoots

roots = nextRoots

move i towards origin

draw line segments from roots to vertex

This draws the entire curve using purely line segments.

**V. Conclusion**

The aim of this project was to demonstrate the performance of these transformations on the given objects using the Java programming language. Given the final product, this objective was fulfilled. The team was able to draw each object and perform all appropriate transformations using matrices in the Java programming language.

**VI. References**

Rotation of curves in two dimensions. (n.d.). Retrieved October 9, 2015, from Department of Mechanical

and Aerospace Engineering: University of Florida: http://www2.mae.ufl.edu/~uhk/ROTATION

.pdf

Transformation matrix. (n.d.). Retrieved October 9, 2015, In Wikipedia: https://en.wikipedia.org/wiki

/Transformation\_matrix

**Appendix A: Objects and their Transformations**

**Point**

* Translate by

**Vector**

Translate by

Rotate by d degrees

Scale by factor r

Reflect

X-axis

Y-axis

Shear by d degrees

**LINE SEGMENT - apply the above to endpoints**

**POLYGON - apply the above to all vertices**

**PARABOLA**

**Vertical**

Translate by

Add to

Scale by r

Multiply to

Rotate

90 clockwise

Vertex

Magnitude:

Opening Horizontal

180

Vertex:

Magnitude:

Opening Horizontal

90 counter clockwise

Vertex:

Magnitude:

Opening Horizontal

Reflect

X-axis

Vertex:

Magnitude:

Opening Vertical

Y-axis

Vertex:

Magnitude:

Opening Vertical

**Horizontal**

Translate by

Add to

Scale by r

Multiply to

Rotate

90 clockwise

Vertex:

Magnitude:

Opening Vertical

180

Vertex:

Magnitude:

Opening Horizontal

90 counter clockwise

Vertex:

Magnitude:

Opening Vertical

Reflect

X-axis

Vertex:

Magnitude:

Y-axis

Vertex:

Magnitude:

Opening Horizontal

**ELLIPSE**

Translate by

Add to

Scale by r

Multiply to

Rotate

90 clockwise

Center :

h.d. = b

v.d. = a

180

Center :

h.d. = a

v.d. = b

90 counter clockwise

Center :

h.d. = b

v.d. = a

Reflect

x-axis

Center :

h.d. = a

v.d. = b

y-axis

Center :

h.d. = a

v.d. = b

**HYPERBOLA**

**Vertical**

Translate

add [a b] to [h k]

scale

multiply [h.d. v.d.] by r

rotate

90 clockwise

Center :

h.d. = b

v.d. = a

opening vertical

180

Center :

h.d. = a

v.d. = b

opening vertical

90 counter clockwise

Center :

h.d. = b

v.d. = a

opening vertical

Reflect

x axis

Center :

h.d. = a

v.d. = b

opening vertical

y axis

Center :

h.d. = a

v.d. = b

opening vertical

horizontal

Translate

add [a b] to [h k]

scale

multiply [h.d. v.d.] by r

rotate

90 clockwise

Center :

h.d. = b

v.d. = a

opening vertical

180

Center :

h.d. = a

v.d. = b

opening horizontal

90 counter clockwise

Center :

h.d. = b

v.d. = a

opening vertical

Reflect

x axis

Center :

h.d. = a

v.d. = b

opening horizontal

y axis

Center :

h.d. = a

v.d. = b

opening horizontal